Universidad de las Fuerzas Armadas ESPE
Technologies de software para electrónica
Producto de Unidad

Comunicacion Serial

10 de mayo de 2019

Socasi Bryan, Viera Katherine, Yanez Danilo
basocasi@espe.edu.ec
kaviera1@espe.edu.ec
djyanez1@espe.edu.ec

Índice general

1. Introducción

La evolución de la tecnología en las últimas décadas ha permitido generar nuevos protocolos y dispositivos para la transmisión de datos entre una computadora y un periférico, en sus inicios alrededor de los años 60 esta transmisión de información se daba a través de puertos seriales enviando datos bit a bit con tiempo de espera prolongado, después en 1998 fue reemplazada por puertos USB's que tenían mejores características y mayor eficiencia. Actualmente los individuos no tienen pleno conocimiento sobre los fundamentos o la estructura base que permitió que la comunicación pueda darse como se la conoce ahora, sin embargo, es muy importante entender los inicios en esta área especialmente para los estudiantes de ingeniería en electrónica, pues de esta manera se logrará entender la diversidad que ha existido a lo largo del tiempo y cómo funciona cada "cosa" que utilizamos. Relacionando los temas aprendidos en clase se desarrolla un trabajo de investigación que involucra la comunicación serial entre dos PC a través de una interfaz gráfica desarrollada con lenguaje de programación Java, controlada por un arduino.

2. Objetivos

2.1. Objetivo general

Desarrollar una aplicación en comunicación serial mediante el uso de Arduino y Java para poder controlar sensores y actuadores, a través de dos computadoras, utilizando programación en Java.

2.2. Objetivos específicos

- Conocer el debido funcionamiento de la comunicación de puertos seriales.
- Diseñar un programa que permita crear un programa de manera eficaz y visual.
- Realizar pruebas, del funcionamiento de la aplicación mediante la conexión de la Comunicación Serial de los sensores y actuadores.

3. Estado del arte

3.1. Java

En (Cisco systems, 2016) se indica qué es Java y su programación orientada a objetos, elementos del lenguaje Java, operadores y estructuras de control del lenguaje Java, fundamentos de definición y uso de clases, sistema, cadenas, cadena de buffer, math y otras clases de envoltura, arrays, clases y herencia, cómo comprender y usar los paquetes, creación de interfaces gráficas de usuario con AWT, applets y gráficos, excepciones, archivos, flujos, entrada y salida, colecciones, hilos.
En (Wielenga, 2015) se indicó como fue el comienzo de la plataforma Neatbeans IDE para programar códigos Java, de esta manera se sabe que es un entorno de desarrollo gratuito y de código abierto que permite el uso de un amplio rango de tecnologías de desarrollo tanto para escritorio, como aplicaciones Web, o para dispositivos móviles. Da soporte a las siguientes tecnologías, entre otras: Java, PHP, Groovy, C/C++, HTML5. Además, puede instalarse en varios sistemas operativos: Windows, Linux, Mac OS .

3.2. Arduino

En (Albatish, Mosa, Abu , 2018) se encontró información sobre cómo enfrentar alguna dificultad al tratar con la plataforma Arduino mediante la descripción del diseño de un sistema de tutoría inteligente basado en escritorio. La idea principal de este sistema es una introducción sistemática al concepto de plataforma Arduino. El sistema muestra las placas de circuito de Arduino y un entorno de desarrollo de fuente abierta y una biblioteca para escribir código para controlar el tema de la placa de la plataforma Arduino. Lo que nos indica en (Pan Zhu, 2018) es que un sensor es ün dispositivo que detecta una variación en la energía de entrada para producir una variación en otra o la misma forma de energía". Por tanto, implementarlos con arduino, es una manera muy interesante de realizar este tipo de proyectos, de esta manera, en este artículo se describe aspectos importantes de su uso.

3.3. Comunicación Serial

En el artículo de (Padrón, Prieto, Herrera, Velazquez, 2018) se indica una aplicación importante con la comunicación serial pues debido a la naturaleza inherente a las comunicaciones móviles, diversos factores influyen en la calidad de los servicios que se desarrollan para los dispositivos de enlace cuando son fabricados, en los cuales no se encuentran técnicas y procedimientos de seguridad implementados en comparación con dispositivos fijos o de escritorio actuales. En este trabajo se presenta la implementación del algoritmo criptográfico por bloques .Advanced Encryption Standard"(AES) de 128 bits, para una comunicación serial entre dos tarjetas Spartan 3E fabricadas por Xilinx. Este algoritmo público según la FIPS-197 es el estándar para muchas aplicaciones en Seguridad Informática tanto en Software como Hardware. Este método criptográfico sigue siendo un mecanismo para implementar los servicios de seguridad recomendados en la ISO 7498-2.

4. Marco teórico

4.1. Arduino Uno

Arduino UNO es una placa electrónica basada en el microcontrolador ATmega 328. Cuenta con 14 pines digitales que pueden ser entradas o salidas de los cuales 6 pueden ser usados como salidas PWM, 6 entradas analógicas, un cristal oscilador de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP y un botón de reinicio. La placa electrónica

tiene todo lo necesario para proporcionar soporte al microcontrolador, basta con conectar la placa a una computadora con un cable USB o a un adaptador AC-DC o a una batería para empezar.

Fig 1. Arduino Uno

La tarjeta Arduino UNO SMD difiere de todas las placas anteriores ya que no utiliza el chip FTDI USB serie. Por el contrario, cuenta con el ATmega16U2 programado como un conversor de USB a serie. “Uno” significa uno en italiano y se llama así por motivos de la próxima versión de Arduino 1.0. La tarjeta Uno y la versión 1.0 serán la versión de referencia de Arduino para seguir adelante. La tarjeta Uno es la última en una serie de placas USB de Arduino, y el modelo de referencia para la plataforma Arduino. (Crespo, 2016)

4.2. Definición de pines

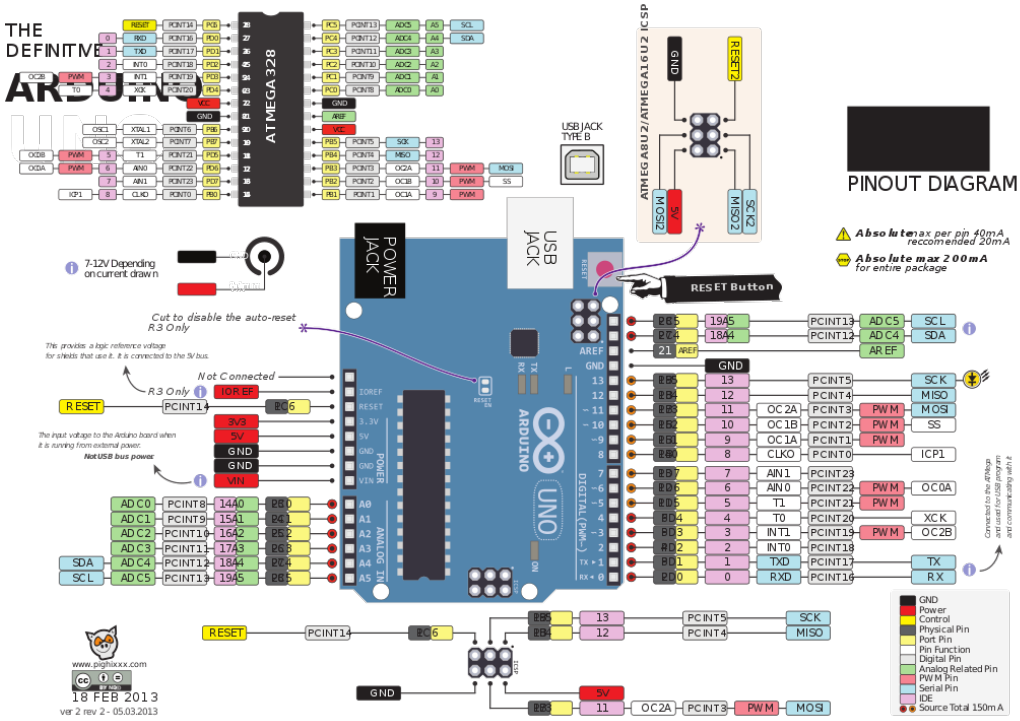


Fig 1. Definición de pines Arduino Uno

4.2.1. Especificaciones Técnicas

Cuadro 1: Especificaciones de la tarjeta Arduino UNO SMD parte 1

Microcontrolador	ATmega328
Voltaje de Operación	5V
Voltaje de alimentación (recomendado)	7-12V
Voltaje de alimentación (limite)	6-20V
Pines Digitales E/S	14(6 proveen salida PWM)
Pines entradas análogas	6
Corriente DC por pin E/S	40mA
Voltaje de Corriente DC por el pin de 3.3V	50mA

Cuadro 2: Especificaciones de la tarjeta Arduino UNO SMD parte 2

Memoria Flash	32KB de los cuales 0.5KB son usados por el gestor de arranque
SRAM	2KB (ATmeta328P)
EEPROM	16MHz
Reloj	16MHz
LED BUILTIN	13
Longitud	53.4 mm
Ancho	53.4 mm
Peso	25 g

4.3. Java

Es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web. El lenguaje de programación Java

fue originalmente desarrollado por James Gosling de Sun Microsystems publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales, y librerías de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath. (Itcea, 2017)

Fig 2. Lenguaje de Programacion JAVA

4.4. API Java Comunication

La API de Java Communications es una extensión de Java que facilita el desarrollo de aplicaciones de comunicaciones independientes de plataforma para tecnologías como Smart Cards, sistemas integrados y dispositivos de punto de venta, dispositivos de servicios financieros, fax, módems, terminales de pantalla y equipos robóticos. La API de comunicaciones de Java también conocida como javax.comm permite a las aplicaciones el acceso al hardware RS-232 (puertos serie) y un acceso limitado a IEEE-1284 (puertos paralelos), modo SPP. (Itcea, 2017)

4.5. Características serial API

- Enumeración de puertos (mapeo de puertos configurables por el administrador y el usuario)

- Configuración del puerto (velocidad en baudios, velocidad, bits de parada, paridad)
- Acceso a señales estándar DTR, CD, CTS, RTS y DSR EIA232
- Transferencia de datos a través de puertos RS-232
- Opciones de control de flujo de hardware y software.
- Control de umbral de buffer de recepción.
- Opción de evento asincrónico para la notificación de:
    - Datos disponibles en un puerto RS232.
    - Cambios de nivel de línea de hardware de puerto.
    - Cambios en la propiedad de puerto de una solo JVM.

4.6. Sensores

Es un concepto genérico que hace referencia a diferentes tipos de sensores, esto se entiende tanto las unidades que emite una señal analógica, como las unidades que emite una señal binaria (encendido o apagado).

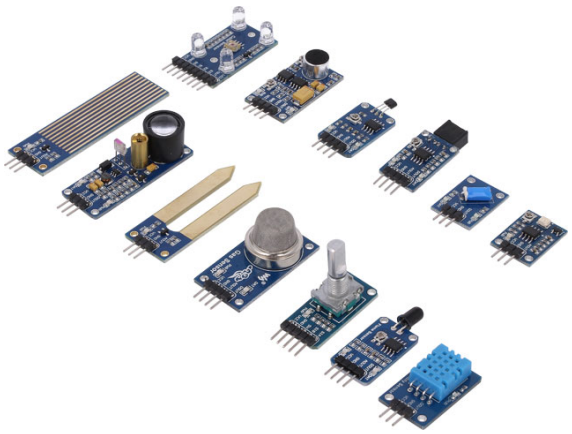


Fig 3. Sensores Electrónicos.

Convierte una magnitud física en una magnitud eléctrica, existe una gama de diferentes productos de sensores para diferentes magnitudes físicas.

4.6.1. Sensor de temperatura

El DHT11 es un sensor de temperatura y humedad el cual dispone de una salida digital calibrada. Su tecnología garantiza la alta fiabilidad y una excelente estabilidad a largo plazo. Es compatible con la tecnología Arduino, PIC, AVR, COP, DSP, STM32, etc.

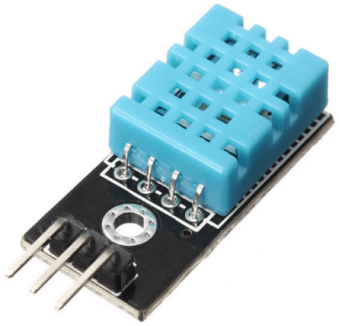


Fig 4. Sensor de Temperatura

Características:

- Compatible con sistemas electrónicos operando entre 3v-5v
- Corriente máxima de 2.5 mA cuando se realiza la conversión.
- Humedad relativa: 0-80
- Temperatura: 0-50°C (±2°C)
- Tiempo de respuesta: 10 segundos
- 4 pines de conexión
- No requiere componentes activos externos.

4.6.2. Sensor de proximidad

Puede detectar objetos metálicos que se acercan al sensor, sin tener contacto físico con los mismos. Los sensores de proximidad inductivos se clasifican más o menos en los siguientes tres tipos, de acuerdo con su principio de funcionamiento: el tipo de oscilación de alta frecuencia que utiliza la inducción electromagnética; el tipo magnético que emplea un imán; y el tipo de capacitancia que aprovecha los cambios en la capacidad eléctrica.

Fig 5. Sensor de Proximidad

Utiliza este sensor ultrasónico en tus proyectos con Arduino o microcontroladores, con el podrás medir la proximidad de algún objeto.

Características:

- Detección de la gama: 3 cm - 4 m
- Mejor en ángulo de 30 grados
- Fuente de alimentación de 5 Vdc
- Transductor Dual

4.6.3. Sensor de CO2

Es un sensor de gas semiconductor sintonizado para detectar monóxido de carbono. Se encuentra en la misma familia de dispositivos que el sensor detector de humo, que mide el cambio en la conductividad de la superficie del dióxido de estaño en presencia de monóxido de carbono. Este sensor tiene una alta sensibilidad y un rápido tiempo de respuesta. La salida del sensor es una resistencia analógica.

Fig 6. Sensor de CO2.

Cuadro 3: Caracteristicas

Zona de detección	10 a 1000ppmm
Voltaje de operación	5V DC
Resistencia de carga	regulable
Sensibilidad	Mayor al 3 %
Tiempo de respuesta	Menos 1s
Resistencia al calentamiento	33 Ohm
Tiempo de calentamiento	60s (alto) 90s (bajo)
Condiciones: Temperatura ambiente	-20C a + 50C
Humedad	95 % RH
Contenido de oxígeno	21 %

5. Diagramas

5.1. Diagrama de bloques

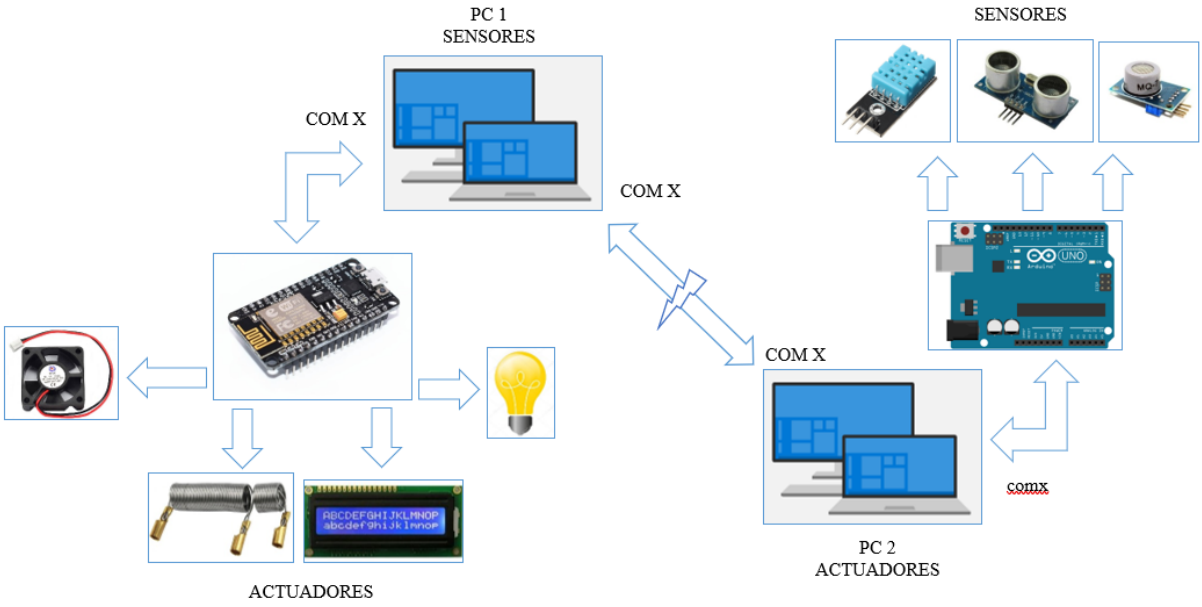


Fig 8. Diagrama de bloques de la Comunicación Serial

5.2. Diagrama UML Sensores

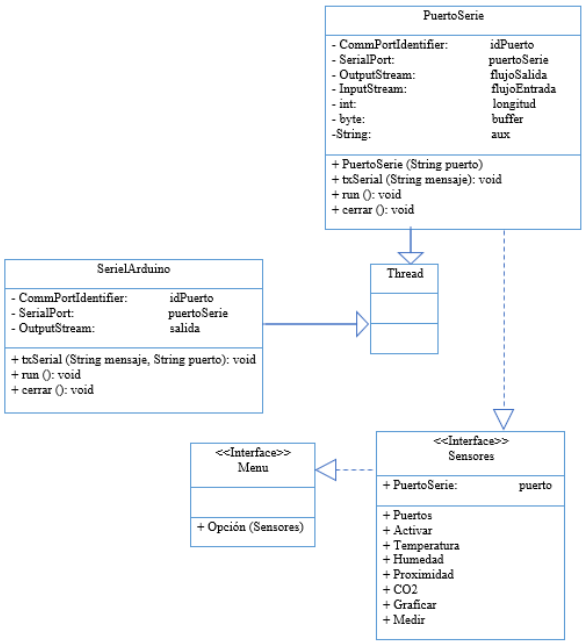


Fig 9. Diagrama UML para los Sensores

5.3. Diagrama UML Actuadores

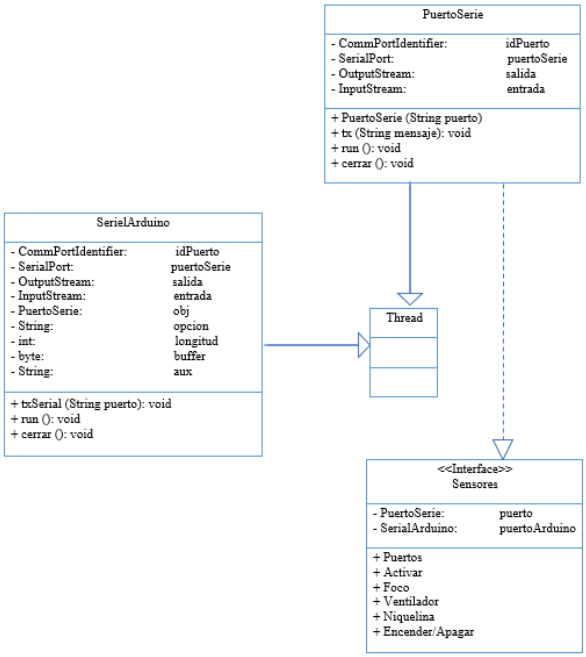
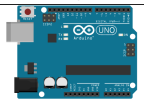


Fig 10. Diagrama UML para los Actuadores

6. Lista de componentes

Cuadro 4: Lista de componentes

Fotografía	Nombre
	Tarjeta de desarrollo Arduino
	Tarjeta de desarrollo ESP8266
	Ventilador
	Niquelina
	Foco
	LCD
	Sensor de temperatura DHT11
	Sensor de proximidad
	Sensor de CO2 MQ7

7. EXPLICACIÓN DEL CÓDIGO FUENTE

7.1. Comunicación Serial

Usaremos el API Java Communications (COMM) que nos permitirá manejar el puerto serie, y así comunicarnos con el otro computador. También usaremos cables adaptadores USB a DB9 y el más importante, un cable DB9 cruzado para que el envió de bits se realice

correctamente. Para usar el api de comunicaciones de java necesitaremos de una librería que contiene las clases para manejar los puertos, estos archivos se deben incluir en la carpeta donde se encuentra el Java instalado, siguiendo las siguientes direcciones:

- Comm.jar \jdk\jre\lib\ext
- Win32com.dll \jdk\bin y en \Windows\system32
- Javax.comm.properties \jdk\jre\lib

Para el código necesitaremos importar la librería mencionada anteriormente

import javax.comm.*;

A continuación tenemos que declarar las clases necesarias para manejar los puertos con sus respectivas variables asignadas, hay que tener en cuenta que la comunicación que usaremos es FULL-DUPLEX, ya que enviaremos y recibiremos información entre ambos computadores.

```
public class PuertoSerie extends Thread{
    private CommPortIdentifier idPuerto;
    private SerialPort puertoSerie; //siempr
    private OutputStream flujoSalida;
    private InputStream flujoEntrada;

    private int longitud = -1;
    private byte[] buffer = new byte[1024];
    private String aux; //Para que el mensaj
```

En donde la Clase CommPortIdentifier administrara la comunicación con los puertos, el acceso a estos y determina y están disponibles o ocupados para establecer la comunicación. La clase SerialPort hereda de la clase abstracta CommPort por lo tanto cuenta con sus métodos pero además posee métodos específicos para tratar los puertos Serie. La clase InputStream y OutputStream nos permitirán establecer el puertos ya sea en modo para recibir o para transmitir datos. Después necesitaremos declarar un constructor de la clase el cual va a recibir como información el puerto al cual nos conectaremos. Este puerto debe ser verificado si esta disponible para la comunicación, también debe ser configurado con parámetros como; velocidad, numero de bits, bit de parada y bit de paridad. Y el modo en el que estará el puerto, recibiendo o transmitiendo.

```
public PuertoSerie(String puerto) throws PortInUseException,
        UnsupportedCommOperationException, IOException{
    try {
            idPuerto=CommPortIdentifier.getPortIdentifier(puerto);
            if(idPuerto.isCurrentlyOwned()){
                System.out.println("puerto en uso");
            } else{
                    puertoSerie=(SerialPort) idPuerto.open("comunicacion",1000);
            System.out.println("Puerto Abierto");
            puertoSerie.setSerialPortParams(115200, SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
            System.out.println("Puerto configurado");
            flujoSalida=puertoSerie.getOutputStream();
            System.out.println("Flujo de salida configurado");
            flujoEntrada=puertoSerie.getInputStream();
            System.out.println("Flujo de entrada configurado");
            this.start();
            }
        } catch (NoSuchPortException ex) {
            Logger.getLogger(PuertoSerie.class.getName()).log(Level.SEVERE, null, ex);
        }
}
```

Para la transmisión de información, declararemos un método con un argumento para transmitir el mensaje deseado, nos ayudaremos de un método .write() que posee la clase SerialPort.

```
public void txSerial(String mensaje){
    try {
        flujoSalida.write(mensaje.getBytes());
        System.out.println(mensaje);
    } catch (IOException ex) {
        Logger.getLogger(PuertoSerie.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Ahora para recibir datos usaremos un método run() de la clase derivada de Thread que nos permitirá crear un subproceso en el programa y con esto poner al puerto en constante “escucha”, por lo que necesitaremos crear un hilo hacia el constructor que contiene la información del puerto usado.

```
public void run() {
    System.out.println("hilo iniciado");
    while (true) {
        try {
            if((longitud=flujoEntrada.read(buffer))>-1){

                aux = new String(buffer,0,longitud);
                System.out.println(aux);
                Sensores.lblRecibir.setText(aux);
                    //CO2.lblCO2.setText(aux);
            }
        } catch (IOException ex) {
            Logger.getLogger(PuertoSerie.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
```

Como usaremos varios sensores y actuares es necesario comunicarse con estos a través del puerto y luego cerrar dicha comunicación, ya que si esta quedara abierta, seria imposible comunicarse con varios sensores o actuadores a la vez, por lo que es necesario crear un método para cerrar los puertos usados.

```
public void cerrar() throws IOException{
    flujoSalida.close();
    flujoEntrada.close();
    puertoSerie.close();
}
```

Ahora necesitamos crear un Jframe para el control de esta clase, para este proyecto es necesario dos GUIs una que controlara los sensores y otra para controlar los actuadores. Básicamente necesitamos colocar un ComboBox con el nombre de los puertos y un botón para activar la comunicación. En el botón agregaremos un evento para que realice una acción al momento de ser presionado, esto nos permitirá escribir el código para activar la comunicación, para lo cual primero debemos crear un objeto de la clase que contiene los parámetros del Puerto serie y luego llamar los métodos necesarios para iniciar la comunicación, llevando la información obtenida del ComboBox y el mensaje deseado al momento de presionar el botón.

```
public class Sensores extends javax.swing.JFrame {
    private PuertoSerie puerto;
    //private SerialArduino puertoArduino;
    SerialArduino puertoArduino=new SerialArduino();
```

```
private void btnPCActionPerformed(java.awt.event.ActionEvent evt) {
try {
        puerto=new PuertoSerie(String.valueOf(cbPuertoPC.getSelectedItem()));
    } catch (PortInUseException ex) {
        Logger.getLogger(Sensores.class.getName()).log(Level.SEVERE, null, ex);
    } catch (UnsupportedCommOperationException ex) {
        Logger.getLogger(Sensores.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Sensores.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

7.2. Comunicación con los actuadores

Como es una comunicación cruzada, es necesario que en el software que controla los sensores tenga la capacidad de recibir los datos para controlar los actuadores, para ello se uso una clase muy similar a la de la Comunicación con el puerto serie, con la diferencia que se puso al puerto correspondiente de los actuadores en un subproceso de constante escritura de los datos recibidos por el otro computador.

```
public void run(){
    while(true){
        try {
            salida.write(lblRecibir.getText().getBytes());
        } catch (IOException ex) {
            Logger.getLogger(SerialArduino.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
}
```

7.3. Comunicación con los sensores

Para realizar la comunicación con los sensores debemos definir un nuevo se debe abrir y configurar un nuevo puerto en el que estará conectado nuestro arduino.

```
public void txSerial(String dato, String puerto) throws NoSuchPortException,
        PortInUseException, IOException, UnsupportedCommOperationException
{
   idPuerto=CommPortIdentifier.getPortIdentifier(puerto);
    System.out.println("Puerto "+puerto+" de arduino identificado");
    if(idPuerto.isCurrentlyOwned()){
        System.out.println("Puerto en uso");
    }else{
        puertoSerie=(SerialPort) idPuerto.open("Puerto", 1000);
        System.out.println("Puerto Abierto");
        puertoSerie.setSerialPortParams(9600, SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
        System.out.println("Puerto Configurado");
        salida=puertoSerie.getOutputStream();
        System.out.println("Flujo de salida configurado");
        entrada=puertoSerie.getInputStream();
        System.out.println("Flujo de entrada configurado");
        //opcion=dato;

        this.start();
    }
}
```

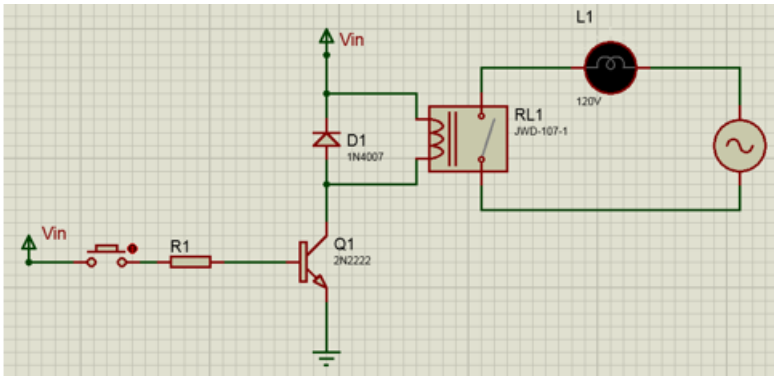
A continuación en el bucle infinito se procede a enviar un caracter al arduino de tal manera que en nuestro código arduino reconoce dicho caracter y retorna el valor que esta midiendo el sensor seleccionado.

```
public void run(){

    System.out.println("hilo iniciado");
    while (true) {
        try {
             salida.write(lblRecibir.getText().getBytes());
             //puerto.tx(lblRecibirArduino.getText());
            if((longitud=entrada.read(buffer))>-1){
                aux = new String(buffer,0,longitud);
                System.out.println(aux);
                Actuadores.lblRecibirArduino.setText(aux);
                //puerto.tx(lblRecibirArduino.getText());
                comunicacion.tx(lblRecibirArduino.getText());
            }

        } catch (IOException ex) {
            Logger.getLogger(SerialArduino.class.getName()).log(Level.SEVERE, null, ex);
        }



    }
}
```

7.4. Programación en Arduino

7.4.1. Actuadores

Para los actuadores se uso una tarjeta ESP8266 la cual se puede programar en el IDE de Arduino, necesitamos manejar los pines digitales que posee la tarjeta, enviando un 0 o 1 logico para asi activar los controladores, estos controladores son: foco, niquelina, ventilador y un LCD. El foco necesita un circuito de control y uno de potencia, además de que la niquelina como el ventilador necesitan mas voltaje que el que entrega la tarjeta, por lo que usaremos transistores para amplificar la señal lógica que recibimos y de relés para controlar la parte de potencia, siguiendo el siguiente diagrama.



Como se menciono antes, necesitaremos controlar un LCD para lo cual nos ayudamos de un modulo I2C por lo necesitaremos importar una librería al IDE de Arduino. Luego procedemos a declarar los pines de la tarjeta en variables relacionadas con los actuadores a los que estarán conectados. En el void setup() empezaremos a declarar el modo en el que trabajaran los pines definidos anteriormente, en este caso todos serán OUTPUT (salida) y también se procedera a inicializar el lcd. En el void loop() usaremos la condicion if-else, ya que en la comunicación en serie creada anteriormente, se transmitirán “caracteres” al momento de presionar los botones para activar o desactivar los dispositivos, por lo que necesitaremos comparar dichos caracteres recibidos con variables auxiliares. La lógica es que si estos caracteres recibidos son iguales a los comparados entrara en el laso if y ejecutara el comando para enviar un HIGH al pin seleccionado, y se seguirá esta lógica para todos los actuadores, pero para el lcd necesitaremos enviar el mensaje deseado, por lo que usaremos el comando lcd.print

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h> //configurar puertos base

const int foco=14;
const int ventilador=12;
const int niquelina=13;


LiquidCrystal_I2C lcd(0x27, 16, 2); //protocolo 0x27, 16x2

void setup() {
pinMode(foco, OUTPUT);
pinMode(ventilador, OUTPUT);
pinMode(niquelina, OUTPUT);
pinMode(2, OUTPUT);
Wire.begin(D2, D1); // D1 = SCL | D2 = SDA
lcd.begin(); // Iniciamos el LCD
lcd.backlight(); // Prendemos la Iluminacion del LCD
lcd.clear();
Serial.begin(115200);
}
```

```
void loop() {
if(Serial.available()>0){
  int input=Serial.read();
//foco
  if(input=='0'){
    digitalWrite(foco, HIGH);
  }
  else if(input=='1'){
    digitalWrite(foco, LOW);
  }
//ventilador
    if(input=='2'){
    digitalWrite(ventilador, HIGH);
  }
  else if(input=='3'){
    digitalWrite(ventilador, LOW);
  }
//niquelina
    if(input=='4'){
    digitalWrite(niquelina, HIGH);
  }
  else if(input=='5'){
    digitalWrite(niquelina, LOW);
  }
```

7.4.2. Sensores

Para los sensores se usó una tarjeta Arduino UNO, y de igual manera necesitaremos controlar los pines digitales y también los analógicos para recibir los datos obtenidos por

los sensores. Se procede a declarar los pines de la tarjeta en variables relacionadas con los sensores a los que estarán conectados. En el void setup() empezaremos a declarar el modo en que trabajaran los pines definidos anteriormente, en este caso necesitaremos que sean INPUT (entrada) y un OUTPUT (salida) En el void loop() seguiremos la misma lógica que en los actuadores, vamos a comparar los caracteres recibidos por la comunicación serial y las compararemos mediante variables auxiliares para así entrar al laso if-else que contendrán los métodos necesarios para el funcionamiento de los sensores.

```
#include <DHT.h>

const int sensorDHT=A0;
int temp,humedad;
DHT dht (sensorDHT,DHT11);

const int echo = 2;
const int trigger = 3;
const int co2D=4;
const int co2A=A1;
int valor;
int limite;
long duracion,distancia;
int i=0;

void setup() {
  Serial.begin(9600);
  dht.begin(); //Iniciamos nuestro sensor DHT11
  pinMode(trigger,OUTPUT);
  pinMode(echo,INPUT);
  pinMode(temp,INPUT);
  pinMode(co2D,INPUT);
  pinMode(co2A,INPUT);
}
```

```
void loop() {
  delay(1000);
    char selector=Serial.read();
  for(int i=0;i<10;i++){
    if(selector=='A'){
   Temperatura();
  }
  if(selector=='B'){
    Humedad();
  }
  if(selector=='C'){
    Proximidad();
  }
  if(selector=='D'){
    CO2();
  }
```

```
void Temperatura(){
  temp= dht.readTemperature(); //Permite leer la temperatura.
  Serial.print(temp);
  Serial.print("°C"); //Tempertura: 29°C
  delay(3000);

}

void Humedad(){
  humedad= dht.readHumidity(); //Funcion incluida en la libreria. Permite leer la humedad.
  Serial.print(humedad);
  Serial.println("%");
  delay(3000);
}

void Proximidad(){
  digitalWrite(trigger, LOW);
  delayMicroseconds(2);
  digitalWrite(trigger, HIGH);   // genera el pulso de triger por 10ms
  delayMicroseconds(10);
  digitalWrite(trigger, LOW);
```

8. CONCLUSIONES

- La comunicación de puerto serial ha tenido una breve evolución en las últimas décadas, a través del estado del arte se comprobó su aplicación y funcionamiento con la transmisión de datos, en la creación del sistema de comunicación serial presentado en este informe, se procedió a la conexión entre dos computadoras con el Arduino y el uso del ESP8266, enviando datos para realizar la comunicación serial tanto para los sensores y los actuadores, por medio de una consola de java y Arduino.

- El Arduino se puede utilizar para diferentes ámbitos como desarrollar elementos autónomos, conectándose a dispositivos e interactuar tanto con el hardware como con el software; es una plataforma electrónica que en este trabajo nos permitió la interacción con una interfaz gráfica desarrollada con lenguaje de programación Java, con la finalidad de poder tener una interacción más fácil del Arduino con el computador.

- Se logro cumplir con cada uno de los objetivos, y entender el desarrollo de todos los programas, implementados en dos computadoras, con el debido funcionamiento de los Sensores y Actuadores.

9. RECOMENDACIONES

- Se debe considerar que los pines en los que se conecta cada uno de los sensores y actuadores, que están configurados para funcionar en un orden específico, para obtener los datos requeríos en el Producto de unidad.
- Se debe tener en consideración que los paquetes utilizados en Java, no son propios de ahí, pero se debe descargar las librerias, y el funcionamiento de código nos permita trabajar de manera eficaz en los programas implementados..
- Conectar correctamente nuestros elementos, tanto a la tarjeta de Desarrollo del ESP8266 y la tarjeta de desarrollo Arduino UNO, debido a que, aunque realicemos correctamente la programación y la configuración con el servidor, los Actuadores y sensores se encuentran en los pines incorrectos o con la polarización indebida estos no podrán funcionar como esperamos.

Referencias

A., P. (17 de 10 de 2012). twenergy. Recuperado el 04 de 28 de 2018, de https://twenergy.com/a/como-se-genera-la-electricidad-666

Albatish, I., Mosa, M., Abu , S. (01 de 2018). ResearchGate. Obtenido de $323322871_{A}RDUINO_{T}utor_{A}n_{I}ntelligent_{T}utoring_{S}ystem$

BBVA Open4U. (15 de 08 de 2015). BBVA. Obtenido de https://bbvaopen4u.com/es/actualidad/el-internet-de-las-cosas-de-codigo-abierto-plataformas-y-aplicaciones-para

Carlos, C. (12 de 06 de 2015). Galia,FC. Obtenido de http://galia.fc.uaslp.mx/

Cisco systems. (03 de 2016). ResearchGate. Obtenido de https://www.researchgate.net/publication/

Crespo, E. (25 de 09 de 2016). Aprendiendo Arduino. Obtenido de https://aprendiendoarduino.wordpress.com/2016/09/25/que-es-arduino/

Facultad Tecnológica. (02 de 10 de 2014). Universidad Distrital Francisco José de Caldas. Recuperado el 28 de 04 de 2018, de http://gemini.udistrital.edu.co/comunidad/grupos/gispud/Circuitos-II/Capitulo-3/3.2.1.html

García, J. A. (s.f.). AF. Obtenido de AF: http://www.asifunciona.com/fisica/af$_{d}iodos/af_{d}iodos_{6}.htm$

N.O.Sadiku, C. K. (2013). En Fundamentos de Circuitos Eléctricos (pág. 866). México: Mc Graw Hill Education

10. ANEXOS

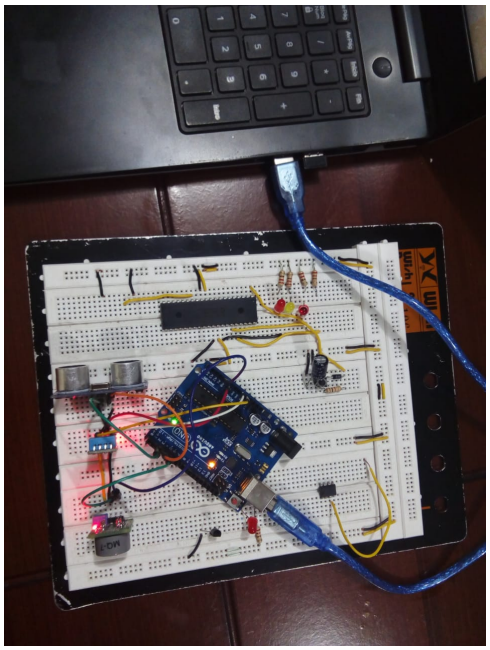


Fig 7. Sensores

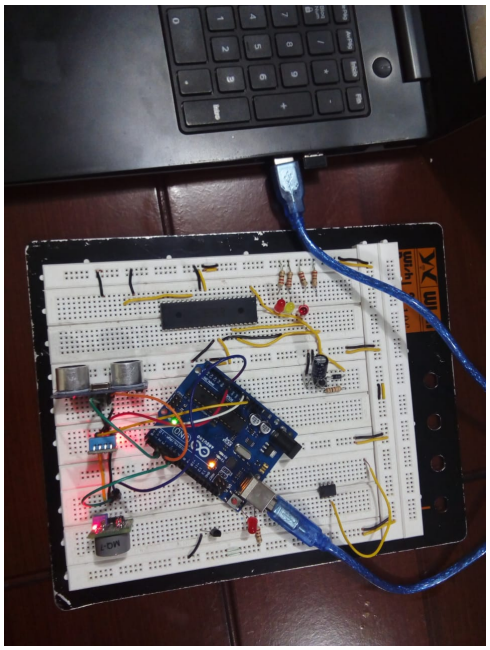


Fig 7. Actuadores