VERDAD, BELLEZA, PROBIDAD

Universidad Autónoma de Tamaulipas

UNIDAD ACADEMICA MULTIDISCIPLINARIA MANTE

REPORTE

PRÁCTICAS

Integrantes:
Avalos Castillo Armando de Jesus

Fortanelly Palomares Guillermo Andres

Hernández Pérez Juan Yahir

Martínez Flores Karla Selene

Vázquez Serrato Jorge Emmanuel

Asignatura:
Programación de Interfaces y Puertos

Grado y Grupo:
6°J

Docente:
López Piña Daniel

INDICE

INTRODUCCIÓN

El presente reporte documenta una serie de prácticas centradas en la lectura de datos desde sensores analógicos mediante un microcontrolador Arduino, y el control de un actuador (LED) a través de comandos enviados por el puerto serie desde una computadora.

Estas prácticas se realizaron utilizando los lenguajes de programación C y Python en los sistemas operativos Linux y Windows, lo que permite comparar enfoques, herramientas y particularidades en distintas plataformas. A lo largo de las actividades, se aplicaron conceptos fundamentales como la comunicación serial, la lectura y conversión de datos analógicos, el control digital, y el desarrollo de interfaces de software capaces de interactuar directamente con el hardware.

Este conjunto de ejercicios sirve como introducción práctica al mundo de los sistemas interactivos y la programación de bajo nivel con alto impacto en proyectos de automatización y control.

PRÁCTICAS

Control de Motores DC con Puente H L298N

Objetivo
Aprender a conectar y controlar la dirección y velocidad de motores de corriente directa (DC) utilizando el módulo de puente H L298N en conjunto con una placa Arduino.

Materiales

- Placa Arduino (Uno, Mega, etc.)
- Módulo Puente H L298N
- 2 Motores DC
- Fuente de alimentación externa (6V–12V, según especificaciones de los motores)
- Cables de conexión (jumpers)
- Computadora con el IDE de Arduino instalado

Esquema de Conexión

1. **Conexión del Módulo L298N:**

- VCC (12V) conectado a la fuente de alimentación externa.
- GND conectado tanto a la fuente externa como al GND del Arduino.
- VCC (5V) conectado a 5V del Arduino (o utilizando el jumper de 5V del módulo).
- OUT1 y OUT2 conectados a los terminales del primer motor DC.
- OUT3 y OUT4 conectados a los terminales del segundo motor DC.
- IN1, IN2, IN3, IN4 conectados a pines digitales del Arduino (por ejemplo: 8, 9, 10 y 11).
- ENA y ENB conectados a pines PWM del Arduino (por ejemplo: 3 y 5) para control de velocidad.

2. **Conexión de la Fuente de Alimentación:**

- La fuente de alimentación externa debe conectarse directamente al módulo L298N.

Código de Control en Arduino

```
// Pines de control para el Motor A
int motorA_IN1 = 8;
int motorA_IN2 = 9;
int motorA_ENA = 3; // Pin PWM
```

```
// Pines de control para el Motor B
int motorB_IN3 = 10;
int motorB_IN4 = 11;
int motorB_ENB = 5; // Pin PWM

void setup() {
  // Configurar pines como salidas
  pinMode(motorA_IN1, OUTPUT);
  pinMode(motorA_IN2, OUTPUT);
  pinMode(motorA_ENA, OUTPUT);

  pinMode(motorB_IN3, OUTPUT);
  pinMode(motorB_IN4, OUTPUT);
  pinMode(motorB_ENB, OUTPUT);
}

void loop() {
  // Motor A hacia adelante, velocidad máxima
  digitalWrite(motorA_IN1, HIGH);
  digitalWrite(motorA_IN2, LOW);
  analogWrite(motorA_ENA, 255);

  // Motor B hacia atrás, velocidad media
  digitalWrite(motorB_IN3, LOW);
  digitalWrite(motorB_IN4, HIGH);
  analogWrite(motorB_ENB, 150);

  delay(2000); // Esperar 2 segundos

  // Motor A hacia atrás, velocidad baja
  digitalWrite(motorA_IN1, LOW);
  digitalWrite(motorA_IN2, HIGH);
  analogWrite(motorA_ENA, 100);

  // Motor B hacia adelante, velocidad máxima
  digitalWrite(motorB_IN3, HIGH);
  digitalWrite(motorB_IN4, LOW);
  analogWrite(motorB_ENB, 255);

  delay(2000); // Esperar 2 segundos

  // Detener ambos motores
  digitalWrite(motorA_IN1, LOW);
  digitalWrite(motorA_IN2, LOW);
  digitalWrite(motorB_IN3, LOW);
  digitalWrite(motorB_IN4, LOW);
  analogWrite(motorA_ENA, 0);
```

```
  analogWrite(motorB_ENB, 0);

  delay(2000); // Esperar 2 segundos
}
```

Procedimiento

1. **Carga del Código:**

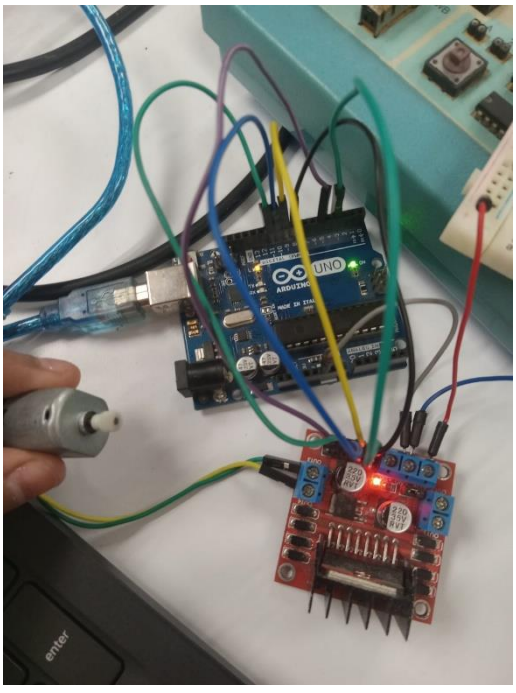
- Se conecta la placa Arduino a la computadora mediante un cable USB.
- Se carga el código de control desde el IDE de Arduino.

2. **Ejecución y Pruebas:**

- Se enciende la fuente de alimentación externa conectada al puente H.
- Se observa cómo los motores cambian de dirección y velocidad según las instrucciones del programa.
- Se puede modificar el valor de analogWrite() para variar la velocidad de los motores, o cambiar las señales digitalWrite() para invertir su dirección.

Resultados

Durante la práctica, los motores giraron correctamente en ambas direcciones y a diferentes velocidades, según las señales enviadas por el Arduino. La variación de velocidad mediante PWM fue evidente y efectiva. También se verificó que el puente H L298N permite manejar sin problema dos motores a la vez, siempre que se utilice una fuente externa adecuada.



Conclusiones

- El módulo puente H L298N permite controlar de forma sencilla y efectiva la dirección y velocidad de motores DC.
- La utilización de PWM para el control de velocidad brinda una solución flexible y precisa.
- Es fundamental utilizar una fuente de alimentación externa, ya que el consumo de corriente de los motores excede la capacidad del Arduino.
- Esta práctica sienta las bases para desarrollar sistemas de robótica móvil o mecanismos motorizados controlados electrónicamente.

Control de Carga de CA con Relé y Arduino

Objetivo

Aprender a conectar y controlar un relé mediante Arduino, con el fin de manejar una carga de corriente alterna (CA) de forma segura y eficiente.

Materiales

- Placa Arduino (Uno, Mega, etc.)
- Módulo de relé (de un canal)
- Foco o lámpara de CA (preferiblemente de baja potencia)
- Portalámparas
- Cable de CA
- Cables de conexión (jumpers)
- Fuente de alimentación de CA (120V o 220V, según la región)
- Computadora con el IDE de Arduino instalado
- Diodo de protección 1N4001 (opcional, pero recomendado)

Advertencia de Seguridad

- La corriente alterna (CA) puede ser peligrosa. Es fundamental tomar todas las precauciones necesarias. En caso de duda, se recomienda buscar la ayuda de un electricista calificado.
- Verifica siempre que la carga conectada esté dentro de los límites de corriente y voltaje especificados por el relé.

Esquema de Conexión

1. **Conexión del Módulo de Relé:**

- VCC del módulo al pin de 5V del Arduino.
- GND del módulo al GND del Arduino.
- IN a un pin digital del Arduino (por ejemplo, pin 7).
- Diodo 1N4001 en paralelo con la bobina del relé (opcional pero recomendado):
- Cátodo (banda gris) conectado al VCC del relé.
- Ánodo al GND del relé.

2. **Conexión de la Carga de CA:**

- Se conecta un cable de CA desde la fuente de alimentación.
- Se corta uno de los conductores de fase del cable de CA.
- El cable cortado se conecta a los terminales COM (común) y NO (normalmente abierto) del relé.
- El portalámparas con la lámpara se conecta en serie con esta línea.

Código de Arduino

```
int relayPin = 7; // Pin conectado al relé

void setup() {
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, LOW); // Inicialmente, el relé está apagado
}

void loop() {
  digitalWrite(relayPin, HIGH); // Enciende el relé
  delay(2000); // Espera 2 segundos

  digitalWrite(relayPin, LOW); // Apaga el relé
  delay(2000); // Espera 2 segundos
}
```

Procedimiento

1. **Carga del Código:**

- Se conecta la placa Arduino a la computadora mediante cable USB.
- Se abre el IDE de Arduino y se carga el código de control.

2. **Encendido y Prueba:**

- Se conecta la fuente de alimentación de corriente alterna.
- Se observa cómo el foco se enciende y apaga cada 2 segundos, de acuerdo con el programa cargado.
- Es posible modificar los valores de delay() para ajustar los intervalos de encendido y apagado de la carga.

Resultados

Durante la ejecución del programa, el relé funcionó como un interruptor electrónico, permitiendo el encendido y apagado de la lámpara de CA en intervalos regulares. La respuesta del relé fue rápida y precisa. El uso del diodo de protección ayudó a evitar posibles picos de tensión inversa que podrían dañar el microcontrolador.

Conclusiones

- El uso de un relé permite controlar cargas de corriente alterna de manera segura mediante señales de bajo voltaje generadas por el Arduino.
- Es indispensable respetar las especificaciones eléctricas del relé para evitar sobrecalentamientos o fallos del sistema.

Control de Microservo con Arduino

Objetivo

Aprender a conectar y controlar un microservo utilizando Arduino, logrando movimientos precisos a diferentes ángulos mediante señales PWM.

Materiales

- Placa Arduino (Uno, Mega, etc.)
- Microservo (SG90 u otro similar)
- Cables de conexión (jumpers)
- Computadora con el IDE de Arduino instalado

Esquema de Conexión

1. **Conexión del Microservo:**

- Cable rojo (VCC) conectado a 5V del Arduino.
- Cable negro o marrón (GND) conectado a GND del Arduino.
- Cable naranja o amarillo (señal) conectado a un pin digital (por ejemplo, pin 9).

Código de Arduino

```
#include <Servo.h>

Servo miServo; // Crea un objeto Servo
int servoPin = 9; // Pin conectado al microservo

void setup() {
  miServo.attach(servoPin); // Asocia el objeto Servo al pin digital
}

void loop() {
  miServo.write(0);     // Mueve el servo a 0 grados
  delay(1000);          // Espera 1 segundo

  miServo.write(90);    // Mueve el servo a 90 grados
  delay(1000);          // Espera 1 segundo

  miServo.write(180);   // Mueve el servo a 180 grados
  delay(1000);          // Espera 1 segundo

  miServo.write(45);    // Mueve el servo a 45 grados
  delay(1000);          // Espera 1 segundo

  miServo.write(135);   // Mueve el servo a 135 grados
```

```
  delay(1000);          // Espera 1 segundo

  miServo.write(0);     // Regresa el servo a 0 grados
  delay(1000);          // Espera 1 segundo
}
```

Procedimiento

1. **Instalación de la Librería:**

- La librería Servo viene preinstalada con el IDE de Arduino, por lo que no es necesario instalarla manualmente.

2. **Carga del Código:**

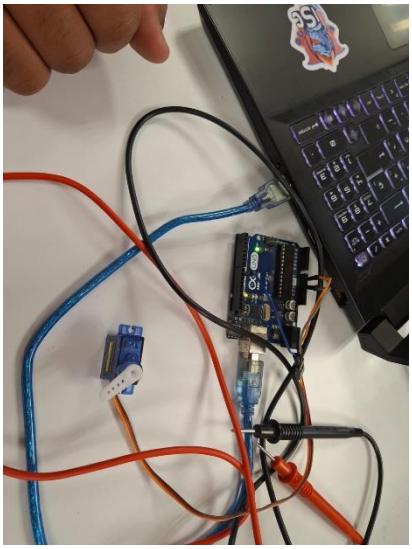
- Conecta la placa Arduino a la computadora mediante el cable USB.
- Abre el IDE de Arduino, copia el código anterior y súbelo a la placa.

3. **Visualización del Movimiento:**

- Una vez cargado el código, observa cómo el microservo se mueve automáticamente a diferentes ángulos según lo programado.
- Puedes modificar los valores en miServo.write() para ajustar los ángulos de movimiento y los delay() para cambiar el tiempo entre cada posición.

Resultados

El microservo ejecutó los movimientos con precisión, desplazándose entre varios ángulos definidos en el programa. Los cambios de posición fueron suaves y estables, lo que demostró el correcto funcionamiento de la señal PWM generada por la librería Servo de Arduino.



Conclusiones

- Los microservos pueden controlarse de forma sencilla mediante la señal PWM generada por el Arduino.
- La librería Servo facilita enormemente el manejo de este tipo de actuadores, permitiendo controlar el ángulo con un solo comando.
- Esta práctica sirve como base para aplicaciones más avanzadas como brazos robóticos, mecanismos automáticos y sistemas de control por sensores.

Aprendizaje

- Conexión eléctrica básica de un microservo con Arduino.
- Uso práctico de la librería Servo para generar señales PWM.
- Control de movimientos precisos y programables en un actuador.

Lectura de Sensor y Control de LED desde Puerto Serie en C (Linux)

Objetivo

Leer datos enviados desde un Arduino mediante el puerto serie en un sistema Linux utilizando un programa en lenguaje C, y controlar el estado de un LED con base en dichos datos.

Materiales

- Computadora con sistema operativo Linux
- Placa Arduino (Uno, Mega, etc.)
- Sensor analógico (ej. sensor de temperatura)
- LED
- Resistencias (si es necesario)
- Cables de conexión (jumpers)
- IDE de Arduino
- Compilador de C (gcc)

Esquema de Conexión

1. **Conexión del Sensor:**

- Sensor conectado al pin analógico A0 del Arduino.

2. **Conexión del LED:**

- Ánodo del LED conectado al pin digital 13 (o cualquier otro pin de salida digital del Arduino).
- Cátodo conectado a GND mediante una resistencia limitadora.

3. **Puerto Serie:**

- Arduino conectado al puerto USB de la computadora (aparece como /dev/ttyACM0 o /dev/ttyUSB0 en Linux).

Código Arduino

```
const int ledPin = 13; // Pin del LED
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 100; // Intervalo de lectura en milisegundos

void setup() {
  Serial.begin(9600);           // Inicializa comunicación serial
  pinMode(ledPin, OUTPUT);      // Configura el pin como salida
}
```

```
void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    int sensorValue = analogRead(A0); // Lectura del sensor
    Serial.println(sensorValue);      // Envío por puerto serie
  }

  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == '1') {
      ledState = HIGH;
    } else if (command == '0') {
      ledState = LOW;
    }
    digitalWrite(ledPin, ledState); // Aplica el comando al LED
  }
}
```

Código en C (Linux)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

int main() {
  int serial_port = open("/dev/ttyACM0", O_RDWR);
  if (serial_port < 0) {
    perror("Error al abrir el puerto serie");
    return 1;
  }

  struct termios tty;
  memset(&tty, 0, sizeof(tty));

  if (tcgetattr(serial_port, &tty) != 0) {
    perror("Error al obtener atributos del puerto serie");
    return 1;
  }

  tty.c_cflag = CS8 | CREAD | CLOCAL;
  tty.c_cc[VMIN] = 1;
```

```
  tty.c_cc[VTIME] = 5;

  cfsetispeed(&tty, B9600);
  cfsetospeed(&tty, B9600);

  if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
    perror("Error al establecer atributos del puerto serie");
    return 1;
  }

  char buffer[256];
  int bytes_read;
  int sensorValue;

  while (1) {
    bytes_read = read(serial_port, buffer, sizeof(buffer) - 1);
    if (bytes_read > 0) {
      buffer[bytes_read] = '\0';
      sensorValue = atoi(buffer);
      printf("Valor del sensor: %d\n", sensorValue);

      if (sensorValue > 500) {
        write(serial_port, "1\n", 2); // Enciende el LED
      } else {
        write(serial_port, "0\n", 2); // Apaga el LED
      }
    }
    usleep(100000); // Espera 100 ms
  }

  close(serial_port);
  return 0;
}
```

Procedimiento

1. **Carga del Código en Arduino:**

- Sube el sketch desde el IDE de Arduino.

2. **Compilación del Programa en C:**

   gcc serial_sensor_led.c -o serial_sensor_led

3. **Ejecución:**

   ./serial_sensor_led

- Si es necesario, ejecuta con privilegios de administrador:

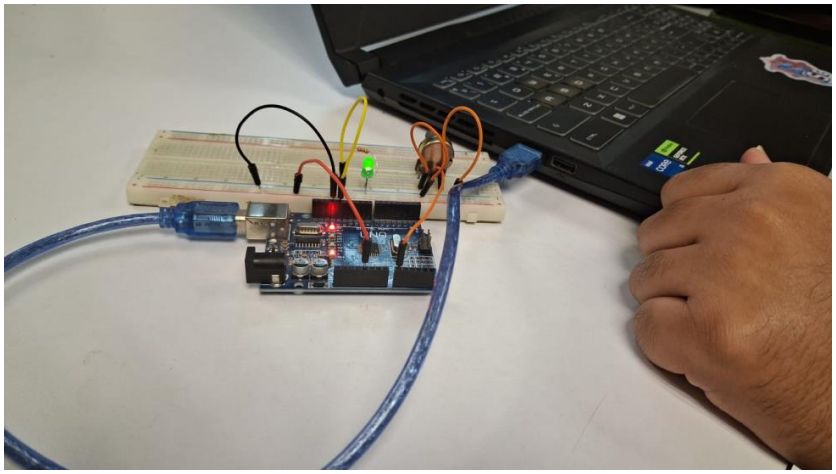
  sudo ./serial_sensor_led

4. **Verificación:**

- El valor del sensor se mostrará por consola.
- El LED se encenderá o apagará dependiendo de si el valor está por encima o por debajo de un umbral (500 en este caso).

Resultados

El sistema logró leer correctamente los datos del sensor desde Arduino y encender o apagar el LED dependiendo del valor leído, utilizando comunicación serial y procesamiento en lenguaje C desde un sistema Linux.



Conclusiones

- Es posible realizar una comunicación bidireccional efectiva entre Arduino y una PC con Linux utilizando el puerto serie.
- El lenguaje C permite analizar en tiempo real datos provenientes de sensores y tomar decisiones como controlar salidas digitales (LED).
- La correcta configuración del puerto serie es esencial para asegurar una comunicación fluida y sin errores.

Puntos Clave

- Es importante conocer el nombre correcto del puerto serie (/dev/ttyACM0, /dev/ttyUSB0, etc.).
- La velocidad de transmisión (baud rate) debe coincidir entre Arduino y el programa en C.
- Pueden requerirse permisos de superusuario para acceder al puerto serie.
- La conversión correcta de cadenas de texto a enteros es fundamental para procesar los datos del sensor.

Lectura de Sensor y Control de LED desde Puerto Serie en Python (Linux)

Objetivo

Implementar una comunicación serie entre Arduino y una computadora con Linux utilizando Python. Se busca leer datos de un sensor conectados al Arduino, procesarlos desde el puerto serie y, en función del valor recibido, encender o apagar un LED.

Materiales

- Placa Arduino (Uno, Mega, etc.)
- Sensor analógico (ej. sensor de temperatura)
- LED
- Resistencias (si es necesario para el LED)
- Cables de conexión (jumpers)
- Computadora con Linux y Python 3 instalado
- IDE de Arduino
- Librería pyserial para Python

Esquema de Conexión

1. Conexión del sensor al Arduino: al pin analógico A0.
2. Conexión del LED: ánodo del LED al pin digital 13 del Arduino (con resistencia limitadora si es necesario), cátodo a GND.
3. Conexión del Arduino al PC: mediante cable USB.

Código Fuente en Arduino

```
const int ledPin = 13;
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 100;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    int sensorValue = analogRead(A0);
    Serial.println(sensorValue);
  }
```

```
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == '1') {
      ledState = HIGH;
    } else if (command == '0') {
      ledState = LOW;
    }
    digitalWrite(ledPin, ledState);
  }
}
```

Código Fuente en Python (Linux)

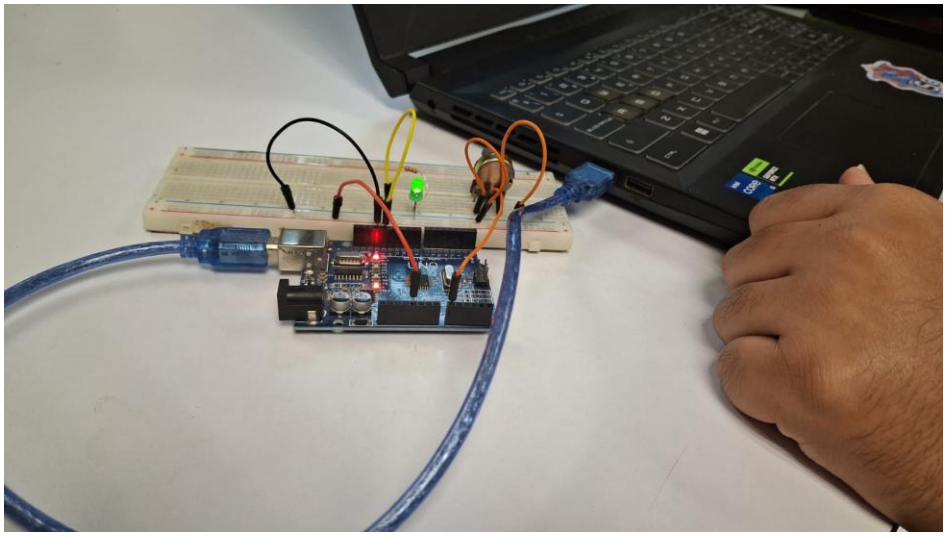
```
import serial
import time

try:
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)  # Cambiar si es necesario
    time.sleep(2)

    while True:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').rstrip()
            try:
                sensor_value = int(line)
                print(f"Valor del sensor: {sensor_value}")

                if sensor_value > 500:
                    ser.write(b'1\n')
                else:
                    ser.write(b'0\n')

            except ValueError:
                print(f"Dato recibido no es un entero: {line}")

        time.sleep(0.1)

except serial.SerialException as e:
    print(f"Error de puerto serie: {e}")
except KeyboardInterrupt:
    print("Programa terminado por el usuario.")
finally:
    if 'ser' in locals() and ser.is_open:
        ser.close()
```

Procedimiento

1. Cargar el código Arduino a través del IDE.
2. Instalar la librería PySerial en Linux con uno de los siguientes comandos:

- sudo apt-get install python3-serial
- pip install pyserial

3. Identificar el puerto serie correspondiente (/dev/ttyACM0, /dev/ttyUSB0, etc.).
4. Ejecutar el programa en Python con el comando:
   python3 serial_sensor_led.py
5. Observar en terminal los valores del sensor y el encendido/apagado del LED según el umbral.



Puntos Clave

- El programa Python realiza una lectura constante del puerto serie y convierte el dato recibido en un número entero.
- Según el valor del sensor, el programa envía un comando ('1' o '0') al Arduino para controlar el LED.
- Se utiliza una estructura try-except en Python para manejo eficiente de errores de lectura y puerto.
- Es fundamental que la velocidad de transmisión coincida entre Arduino y Python (9600 baudios en este caso).
- Si el acceso al puerto serie da error, ejecutar el script con sudo.

Lectura de Sensor y Control de LED desde Puerto Serie en Python (Windows)

Objetivo

Implementar una comunicación serial entre Arduino y una computadora con sistema operativo Windows utilizando Python. Esta práctica permite leer los datos de un sensor conectado al Arduino, procesarlos en Python y controlar un LED dependiendo del valor recibido.

Materiales

- Placa Arduino (Uno, Mega, etc.)
- Sensor analógico (ej. sensor de temperatura)
- LED
- Resistencias (si aplica)
- Cables de conexión (jumpers)
- Computadora con Windows y Python 3 instalado
- IDE de Arduino
- Librería pyserial para Python

Esquema de Conexión

- Sensor analógico al Arduino: conectado al pin A0.
- LED al Arduino: ánodo al pin digital 13 (con resistencia si es necesario), cátodo a GND.
- Arduino al PC: mediante cable USB.

Código Fuente en Arduino

```
const int ledPin = 13;
int ledState = LOW;
unsigned long previousMillis = 0;
const long interval = 100;

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    int sensorValue = analogRead(A0);
    Serial.println(sensorValue);
  }
```

```
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == '1') {
      ledState = HIGH;
    } else if (command == '0') {
      ledState = LOW;
    }
    digitalWrite(ledPin, ledState);
  }
}
```

Código Fuente en Python (Windows)

```
import serial
import time

try:
    ser = serial.Serial('COM3', 9600, timeout=1)  # Cambiar 'COM3' si es necesario
    time.sleep(2)

    while True:
        if ser.in_waiting > 0:
            line = ser.readline().decode('utf-8').rstrip()
            try:
                sensor_value = int(line)
                print(f"Valor del sensor: {sensor_value}")

                if sensor_value > 500:
                    ser.write(b'1\n')
                else:
                    ser.write(b'0\n')

            except ValueError:
                print(f"Dato recibido no es un entero: {line}")

        time.sleep(0.1)

except serial.SerialException as e:
    print(f"Error de puerto serie: {e}")
except KeyboardInterrupt:
    print("Programa terminado por el usuario.")
finally:
    if 'ser' in locals() and ser.is_open:
        ser.close()
```

Procedimiento

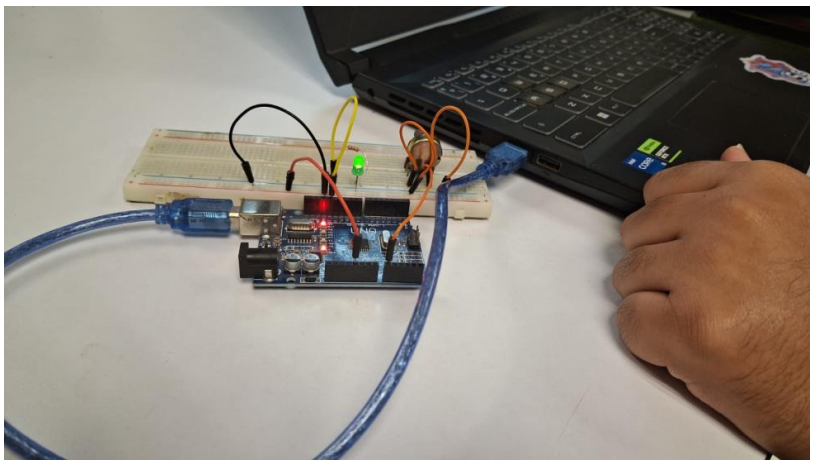
1. **Cargar el programa en Arduino** usando el IDE oficial.
2. **Instalar la librería PySerial** desde el símbolo del sistema:

   nginx
   CopiarEditar
   pip install pyserial

3. **Identificar el puerto serie** del Arduino en el Administrador de Dispositivos (ej. COM3).
4. **Modificar el código Python** si el puerto no es COM3.
5. **Ejecutar el script Python** en la terminal con:

   nginx
   CopiarEditar
   python serial_sensor_led.py

6. **Observar el comportamiento**: si el valor del sensor supera un umbral (500), se enciende el LED; si no, se apaga.



Puntos Clave

- El código Python lee continuamente el puerto serie y convierte los datos recibidos a enteros.
- Se usa un umbral (500) para decidir si se envía un '1' (encender LED) o '0' (apagar LED) al Arduino.
- El Arduino interpreta el comando recibido y actualiza el estado del LED.
- La velocidad de comunicación serial debe coincidir en ambos dispositivos (9600 baudios).
- En Python se implementa manejo de excepciones (try-except) para mayor robustez ante errores de lectura o desconexión del puerto.

CONCLUSIÓN

El desarrollo de las prácticas permitió lograr una comprensión sólida sobre el proceso de comunicación entre un sistema embebido (Arduino) y una computadora mediante el puerto serie. Se demostró cómo leer datos de un sensor y tomar decisiones para controlar un actuador en tiempo real, utilizando tanto C como Python, en entornos Linux y Windows.

Estas experiencias refuerzan la importancia del conocimiento multidisciplinario que combina electrónica, programación y configuración del sistema operativo. Además, se destacó la relevancia del manejo de errores, la detección de puertos serie y el ajuste de parámetros de comunicación como velocidad y formato de datos.

En conjunto, estas prácticas forman una base esencial para abordar proyectos más complejos en automatización.