Materia:

DISEÑO ELECTRÓNICO BASADO EN SISTEMAS EMBEBIDOS

Alumno:

Posadas Pérez Isaac Sayeg

Paniagua Rico Juan Julian

García Azzúa Jorge Roberto

Grado y grupo:

8°G

Profesor:

Garcia Ruiz Alejandro Humberto

Documentación:

Practica 1. Unidad 1

Documentación de la Práctica 1

Introducción

La práctica se centra en la comunicación entre un dispositivo Arduino y un script en Python para la recolección y análisis de datos provenientes de potenciómetros. Se implementan diferentes códigos para establecer la conexión, recibir datos y almacenarlos en un archivo CSV para su posterior análisis. El objetivo es demostrar la interacción entre hardware y software para crear un sistema funcional de adquisición de datos.

Código Arduino

El código que se ejecuta en el Arduino está diseñado para leer los valores de varios potenciómetros y enviar esos datos a través del puerto serial. A continuación se presenta el código correspondiente:

```
const int potPin = A0; // Potentiometer middle pin to A0

const int potPin1 = A1;

const int potPin2 = A2;

const int potPin3 = A3;


void setup() {

  Serial.begin(9600);  // Start serial communication

}


void loop() {
```

```
  int potValue = analogRead(potPin);

  int potValue1 = analogRead(potPin1);

  int potValue2 = analogRead(potPin2);

  int potValue3 = analogRead(potPin3);


  Serial.print(potValue);

  Serial.print(",");

  Serial.print(potValue1);

  Serial.print(",");

  Serial.print(potValue2);

  Serial.print(",");

  Serial.print(potValue3);

  Serial.print(",");


  delay(100); // Small delay for readability

}
```

Este código se encarga de inicializar la comunicación serial a 9600 bps en el `setup()`, y en el `loop()`, lee los valores de cuatro potenciómetros conectados a las entradas analógicas del Arduino (A0 a A3). Los valores se envían como una cadena separada por comas, lo que permite que el script de Python los procese fácilmente.

Módulo `miscUtils.py`

El módulo `miscUtils.py` complementa el proyecto facilitando diversas operaciones comunes en la manipulación de datos y la gestión de puertos seriales. Este módulo incluye las siguientes funciones:

Función `buscarPuertoArduino(vid, pid)`

```
def buscarPuertoArduino(vid=VID,pid=PID):
    puertos = serial.tools.list_ports.comports()
    for puerto in puertos:
        if puerto.vid == vid and puerto.pid == pid:
            return puerto.device
    return None
```

Esta función permite localizar el puerto serial correspondiente a un dispositivo Arduino conectado al sistema. Utiliza los identificadores de vendedor y producto para realizar la búsqueda de manera precisa. Si se encuentra un puerto correspondiente, devuelve el nombre del dispositivo; de lo contrario, devuelve None.

Función `generarLista(min, max, n)`

```
def generarLista(min, max, n):
    return [rd.randint(min, max) for _ in range(n)]
```

Esta función genera una lista de n números enteros aleatorios en un rango definido por los parámetros `min` y `max`.

Función `generarMatriz(max, min, n, m)`

```
def generarMatriz(max, min, n, m):
    return [generarLista(max=max, min=min, n=n) for _ in range(m)]
```

Esta función crea una matriz de tamaño `m x n`, donde cada fila contiene n números enteros aleatorios en el rango definido por `min` y `max`.

Función `print_tournament_results(parent_matrix)`

```
def print_tournament_results(parent_matrix):
    print("Resultados Torneo Binario:")
    print("-" * 50)
    for i in range(len(parent_matrix)):
        min_value = parent_matrix[i, 0, 0]  # Get the minimum value
        formatted_min = f"{min_value:.2f}"
        print(f"Padre {i + 1}:")
        print(f"Valor Minimo: {formatted_min}")
        print(f"Vector: {parent_matrix[i, 1, :]}")
        print("-" * 50)
```

Esta función imprime los resultados de un torneo binario a partir de una matriz de entrada, mostrando el valor mínimo y el vector correspondiente para cada "padre".

Función `print_diccionarios(diccionario)`

```
def print_diccionarios(diccionario):
```

```
for key, value in diccionario.items():
    print(f"Key: {key}, Type: {type(key)}")
    print(f"Valor: {value}, Type: {type(value)}\n")
```

Esta función imprime las claves y valores de un diccionario, así como sus tipos, lo que puede ser útil para depurar y entender la estructura de los datos.

Código Python para la Recepción y Almacenamiento de Datos de Sensores

El script en Python se encarga de buscar el puerto serial al que está conectado el Arduino, establecer la conexión y recibir los datos. A continuación se presenta el código:

```
import serial
import csv
from datetime import datetime
import serial.tools.list_ports
import time


VID = 0x1a86
PID = 0x7523


def buscarPuertoArduino(vid=VID, pid=PID):
    puertos = serial.tools.list_ports.comports()
    for puerto in puertos:
```

```
        if puerto.vid == vid and puerto.pid == pid:
            return puerto.device
    return None


def main():
    puerto = buscarPuertoArduino()
    if not puerto:
        print("No Arduino found")
        return

    try:
        ser = serial.Serial(puerto, 9600, timeout=1)
        time.sleep(10)  # Allow time for serial connection to initialize

        with open('sensor_data.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['Timestamp', 'Temperature', 'Humidity'])

            while True:
                try:
                    line = ser.readline().decode('utf-8', strip=True)
                    if line:
                        data = line.split(',')
                        if len(data) == 4:  # Ensure correct data format
```

```
                    writer.writerow(data)
                    file.flush()
                except Exception as e:
                    print(f"Error reading data: {e}")
                    break

    except serial.SerialException as e:
        print(f"Serial connection error: {e}")

if __name__ == "__main__":
    main()
```

El script comienza buscando el puerto al que está conectado el Arduino mediante la función `buscarPuertoArduino`, que utiliza la biblioteca `serial.tools.list_ports` para listar todos los puertos disponibles y encontrar el que coincide con el ID del vendedor (VID) y el ID del producto (PID) del Arduino. Si no se encuentra el puerto, se muestra un mensaje de error.

Una vez que se establece la conexión serial, el script espera 10 segundos para permitir que la conexión se inicialice antes de abrir un archivo CSV llamado `sensor_data.csv`. El script escribe una cabecera en el archivo y entra en un bucle donde lee continuamente las líneas enviadas por el Arduino. Los datos son procesados y escritos en el archivo CSV. Si se encuentra algún error durante la lectura, se imprime un mensaje y se rompe el bucle.

Código para la Recepción de Señales de Arduino

Además de la recolección de datos, se implementa un código adicional para la recepción y almacenamiento de señales en una matriz:

```
import serial

from Misc.miscUtils import buscarPuertoArduino


puerto_arduino = buscarPuertoArduino()

baud_rate = 9600

matriz = []


def recibirSeñalesArduino(puerto_arduino=puerto_arduino, baud_rate=baud_rate,
matriz=matriz, debug=False):

    try:

        ser = serial.Serial(puerto_arduino, baud_rate, timeout=1)

        print("Conexión establecida")

        while True:

            line = ser.readline().decode("utf-8", errors="ignore").strip().split("-")


            if line:

                if debug:

                    print(f"Valores recibidos: {line}")


                try:

                    parsed_line = []

                    for item in line:

                        if item.isdigit():

                            parsed_line.append(int(item))
```

```
                else:
                    parsed_line.append(item)

            matriz.append(parsed_line)

        except Exception as e:
            print(f"Error al procesar datos: {e}")

except serial.SerialException as e:
    print(f"Error en la conexión serial: {e}")
    return matriz
except KeyboardInterrupt:
    print("\nConexión terminada")
    return matriz
finally:
    if 'ser' in locals() and ser.is_open:
        ser.close()
        print("Puerto serial cerrado")
        return matriz
```

El codigo cumple las siguiente funciones:

Buscar el Puerto Serial: Se utiliza la función `buscarPuertoArduino()` para encontrar el puerto donde está conectado el Arduino. Si no se encuentra, el script imprime un mensaje y finaliza.

Conexión Serial: Si se encuentra el puerto, se establece una conexión serial a 9600 bps. Se incluye un retraso de 10 segundos para permitir que el Arduino se inicialice correctamente antes de comenzar la lectura.

Creación del Archivo CSV: Se abre un archivo CSV llamado `sensor_data.csv` en modo escritura. Se escribe una cabecera en el archivo para identificar los datos que se almacenarán (Timestamp, Temperature, Humidity).

Lectura de Datos: El script entra en un bucle continuo donde lee líneas del puerto serial. Se utiliza `ser.readline()` para leer cada línea, que se espera que contenga los valores enviados por el Arduino. Cada línea se decodifica y se limpia de espacios en blanco.

Validación y Almacenamiento: Los datos leídos se dividen utilizando la coma como delimitador. Se verifica que la longitud de la lista resultante sea 4, lo que asegura que se han recibido todos los datos esperados (4 potenciómetros). Si es así, se escriben en el archivo CSV y se asegura que el archivo se actualice.

Manejo de Errores: Se implementan bloques de manejo de excepciones para capturar y mostrar errores en la conexión serial y en la lectura de datos, lo que permite una identificación rápida de problemas.

Conexión entre los Códigos

La conexión entre el código del Arduino, el módulo `miscUtils.py`, y el script escribir_temp es fundamental para el funcionamiento del sistema de recolección de datos. El Arduino actúa como un dispositivo de adquisición de datos que lee los valores de los potenciómetros y los envía a través del puerto serial. El módulo `miscUtils.py` proporciona la funcionalidad para detectar el puerto serial donde está conectado el Arduino, mientras que el script en Python establece la comunicación con el Arduino, interpreta los datos recibidos y los almacena en un archivo CSV.

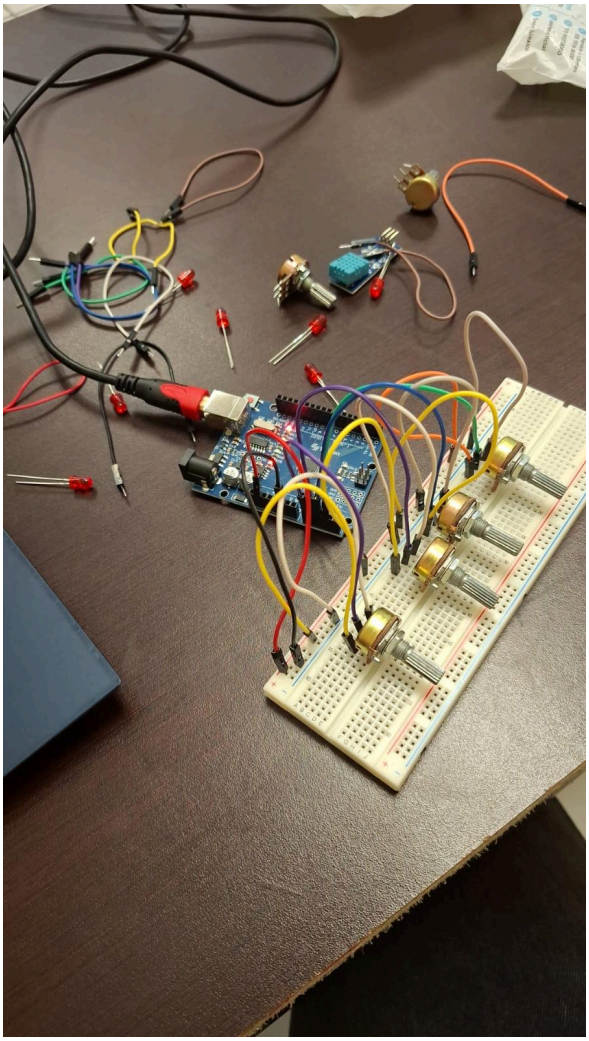
El flujo de datos comienza cuando el Arduino inicia la transmisión de valores de los potenciómetros. Cada 100 ms, los valores son enviados como una cadena de texto. El script en Python escucha el puerto serial, decodifica la información recibida y la divide en elementos individuales, que luego se escriben en el archivo CSV. Este proceso permite una recopilación continua de datos que puede ser utilizada para realizar análisis, visualizaciones o controles de dispositivos basados en los valores leídos.

Aplicaciones y Beneficios

El sistema que se ha desarrollado tiene diversas aplicaciones prácticas. Puede ser utilizado en proyectos de monitoreo ambiental, donde los potenciómetros controlan sensores de temperatura o humedad, proporcionando información valiosa en tiempo real. Además, los datos recolectados pueden ser utilizados para análisis estadísticos, gráficos y visualizaciones, lo que permite una mejor comprensión de los fenómenos medidos.

La implementación de este sistema también demuestra la flexibilidad y capacidad de integración de Arduino y Python. Esta combinación permite crear sistemas más complejos que pueden interactuar con múltiples dispositivos y sensores, facilitando el desarrollo de proyectos innovadores en el campo de la electrónica y la programación.

Fotos de evidencia:



Conclusión

La práctica ilustra cómo podemos establecer una comunicación entre un Arduino y un script de Python para la recolección y análisis de datos. A través del uso de potenciómetros como sensores, se puede interactuar con el entorno y almacenar datos de manera accesible. Este enfoque proporciona una base sólida para el desarrollo de proyectos más complejos, como sistemas de monitoreo, control de dispositivos y visualización de datos en tiempo real. La flexibilidad del Arduino y la facilidad de uso de Python lo convierten en una combinación poderosa para la creación de proyectos de electrónica y programación.