Manual de Uso de Sensores

Este manual presenta una introducción práctica al uso de sensores comunes en proyectos con Arduino. Incluye una descripción general de cada sensor, requerimientos de conexión, bibliotecas de Arduino y Python necesarias para su integración.

Sensores Cubiertos:

1. Sensor de Proximidad IR
2. Sensor Ultrasónico HC-SR04
3. Buzzer (Pasivo y Activo)
4. Sensor de Humedad y Temperatura DHT11/DHT22
5. Sensor de Temperatura LM35
6. Sensor PIR (Movimiento)
7. Sensor de Nivel de Agua / Flotador
8. Sensor de Gas MQ (Serie MQ-x)

1. Sensor de Proximidad IR

Funcionamiento:

Un sensor de proximidad IR emite un haz de luz infrarroja y detecta la reflexión de este haz por un objeto cercano. Algunos tipos tienen una salida digital (presencia/ausencia), mientras que otros proporcionan una salida analógica proporcional a la distancia.

Conexiones (Ejemplo Típico - Salida Digital):

Sensor IR	**Arduino**
VCC	5V
GND	GND
OUT	Un pin digital de Arduino (ej: 2)

Biblioteca Arduino: No requiere.

Consideraciones: Ajuste de sensibilidad mediante potenciómetro.

2. Sensor Ultrasónico HC-SR04

Funcionamiento:

El HC-SR04 emite pulsos de sonido ultrasónico y mide el tiempo que tardan en regresar después de rebotar en un objeto. Utilizando la velocidad del sonido, se calcula la distancia.

Conexiones:

Sensor HC-SR04	**Arduino**
VCC	5V
GND	GND
Trig	Un pin digital de Arduino (ej: 9)
Echo	Otro pin digital de Arduino (ej: 10)

Biblioteca Arduino: NewPing (opcional).

Consideraciones: Requiere cálculos con velocidad del sonido.

3. Buzzer (Pasivo y Activo)

Funcionamiento:

- **Buzzer Activo:** Contiene un circuito oscilador interno y produce un sonido al aplicarle voltaje DC. Es más sencillo de usar para generar tonos únicos.
- **Buzzer Pasivo:** No tiene oscilador interno y requiere una señal AC para producir sonido. Permite generar diferentes tonos y melodías controlando la frecuencia de la señal aplicada.

Conexiones (Buzzer Activo):

Buzzer Activo	**Arduino**
VCC (marcado con '+')	Un pin digital de Arduino con resistencia limitadora (ej: 8 con 220Ω)
GND (marcado con '-')	GND

Conexiones (Buzzer Pasivo):

Buzzer Pasivo	**Arduino**
Un pin	Un pin digital de Arduino con resistencia limitadora (ej: 8 con 220Ω)
Otro pin	GND

Biblioteca Arduino: No requiere, usa tone().

Consideraciones: El pasivo puede generar tonos personalizados.

4. Sensor de Humedad y Temperatura DHT11/DHT22

Funcionamiento:

Los sensores DHT (Digital Humidity and Temperature) proporcionan mediciones digitales de humedad relativa y temperatura. El DHT22 es generalmente más preciso y tiene un rango más amplio que el DHT11.

Conexiones (Ejemplo Típico - 4 Pines):

Sensor DHT	**Arduino**
VCC (Pin 1)	5V
Data (Pin 2)	Un pin digital de Arduino con resistencia pull-up (ej: 4 con 10kΩ)
NC (Pin 3)	No Conectar
GND (Pin 4)	GND

Biblioteca Arduino: DHT.h (de Adafruit).

Consideraciones: DHT22 tiene mejor precisión y rango.

5. Sensor de Temperatura LM35

Funcionamiento:

El LM35 es un sensor de temperatura lineal integrado cuya tensión de salida es directamente proporcional a la temperatura en grados Celsius.

Conexiones:

Sensor LM35	**Arduino**
VCC (Pin 1)	5V
Salida Analógica (Pin 2)	Un pin analógico de Arduino (ej: A1)
GND (Pin 3)	GND

Biblioteca Arduino: No requiere.

Consideraciones: Salida lineal, 10mV/°C.

6. Sensor PIR

Funcionamiento:

Un sensor PIR detecta cambios en los niveles de radiación infrarroja en su campo de visión. Los seres vivos emiten radiación infrarroja, por lo que el sensor puede detectar su movimiento.

Conexiones (Ejemplo Típico - 3 Pines):

Sensor PIR	**Arduino**
VCC	5V
OUT	Un pin digital de Arduino (ej: 3)
GND	GND

Biblioteca Arduino: No requiere.

Consideraciones: Tiempo de calibración al inicio.

7. Sensor de Nivel de Agua / Flotador

Funcionamiento:

Típicamente contiene un interruptor de láminas que se activa o desactiva mediante un imán ubicado en el flotador a medida que el nivel del agua cambia.

Conexiones (Ejemplo Típico - 2 Pines):

Sensor de Nivel	**Arduino**
Un pin	Un pin digital de Arduino con resistencia pull-up o pull-down (ej: 5 con 10kΩ a GND o VCC)
Otro pin	GND (si pull-up) o VCC (si pull-down)

Biblioteca Arduino: No requiere.

Consideraciones: Dependiente del modelo.

8. Sensor de Gas MQ (MQ-2, MQ-3, etc.)

Funcionamiento:

Los sensores de la serie MQ (como MQ-2, MQ-3, MQ-7, etc.) son sensibles a diferentes tipos de gases (humo, alcohol, monóxido de carbono, etc.). Su resistencia interna cambia en presencia del gas objetivo. Generalmente requieren un circuito de calentamiento y proporcionan una salida analógica proporcional a la concentración del gas.

Conexiones (Ejemplo Típico - 4 o 6 Pines):

Sensor MQ-x	**Arduino**
VCC (A o H)	5V
GND (B o H)	GND
Salida Analógica (A o B)	Un pin analógico de Arduino (ej: A2)
Otro pin GND/VCC	Según la hoja de datos del sensor

Biblioteca Arduino: MQUnifiedsensor.h (opcional).

*Nota***:** La estimación precisa de la concentración del gas requiere la hoja de datos específica del sensor MQ que estés utilizando y la curva de sensibilidad proporcionada por el fabricante. La fórmula en el ejemplo es genérica para el MQ-2 (humo) y puede variar significativamente para otros gases.

Uso con Python (Serial) y Bibliotecas/Funciones para Arduino

Para interactuar con los datos de estos sensores desde Python en tiempo real, se utiliza comúnmente la librería pyserial. Después de establecer una conexión serial con el Arduino (configurando el puerto y la velocidad en baudios), Python puede leer los datos que el Arduino envía a través de la función Serial .println() en su código. En Python, se utiliza arduino.readline().decode().strip() para leer una línea de datos, decodificarla de bytes a string y eliminar espacios en blanco. Los datos recibidos suelen ser cadenas de texto que necesitan separadas (separadas y convertidas a números si es necesario).

En el lado de Arduino, las funciones clave para la comunicación serial son:

- Serial.begin(baudrate): Inicializa la comunicación serial a la velocidad especificada.
- Serial.print(data): Envía datos al puerto serial sin añadir un salto de línea al final.
- Serial.println(data): Envía datos al puerto serial añadiendo un salto de línea (\n) al final, lo cual es útil para que Python pueda leer líneas completas.
- analogRead(pin): Lee el valor analógico de un pin (0-1023).
- digitalRead(pin): Lee el estado digital de un pin (HIGH o LOW).
- digitalWrite(pin, value): Escribe un valor digital (HIGH o LOW) en un pin.
- pulseIn(pin, state): Mide la duración de un pulso (en microsegundos) en un pin específico (HIGH o LOW).
- tone(pin, frequency, duration): Genera una señal cuadrada de la frecuencia especificada en un pin durante una duración determinada (para buzzers pasivos).
- noTone(pin): Detiene la generación de la señal en un pin.

Para sensores específicos como el DHT, es común utilizar **librerías de Arduino** desarrolladas por la comunidad (como la librería DHT sensor library de Adafruit) que simplifican la lectura de los datos. Estas librerías proporcionan funciones de alto nivel para inicializar el sensor y obtener las lecturas de humedad y temperatura directamente en las unidades deseadas. Al enviar estos datos a través del puerto serial, Python puede recibirlos y procesarlos para visualización, almacenamiento o control.

SOLID

Los principios SOLID son un conjunto de cinco principios de diseño que pretenden hacer que el diseño del software sea más comprensible, flexible y mantenible. Fueron introducidos por Robert C. Martin.

Los principios son:

- **S**: Principio de responsabilidad única
- **O**: Principio de abierto/cerrado
- **L**: Principio de sustitución de Liskov
- **I**: Principio de segregación de la interfaz
- **D**: Principio de inversión de dependencias

Principio de responsabilidad única

Una clase debe tener una, y sólo una, razón para cambiar, lo que significa que una clase debe tener sólo una responsabilidad. Esto ayuda a mantener cada clase pequeña y enfocada.

Principio de abierto/cerrado

Las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas a la extensión, pero cerradas a la modificación. Esto significa que deberías poder añadir nueva funcionalidad sin cambiar el código existente.

Principio de sustitución de Liskov

Las subclases deben ser sustituibles por sus clases base. Esto significa que cualquier clase que herede de una clase base debe poder utilizarse en lugar de la clase base sin causar ningún comportamiento inesperado.

Principio de segregación de la interfaz

Muchas interfaces específicas del cliente son mejores que una interfaz de propósito general. Esto significa que las clases no deberían verse obligadas a implementar métodos que no utilizan.

Principio de inversión de dependencias

Se debe depender de las abstracciones, no de las concreciones. Esto significa que las clases de alto nivel no deben depender de las clases de bajo nivel, sino que ambas deben depender de las abstracciones.