Proyecto Final: Programación Aplicada

Andres Redondo - 20231005106, David Gaviria - 20231005124

Universidad Distrital Francisco José de Caldas
Ingeniería Electrónica

Palabras clave: MicroPython, ESP32, sensores, salidas digitales.

I. Introducción

Se quiere automatizar un garaje para que las luces, la apertura de la barrera y la medición de la distancia al muro funcionen de forma automática usando salidas digitales. También se usarán LEDs para mostrar cuando hay un vehículo dentro del garaje.

II. Objetivos

Objetivo general

- Diseñar y crear un proyecto en Thonny utilizando MicroPython en la tarjeta ESP32.

Objetivos específicos

- Implementar el sensor HCSR04 en el entorno de Thonny.
- Medir distancias con el HCSR04 e imprimirlas en el puerto serial.
- Mostrar, mediante LEDs, la disponibilidad del garaje, encendiéndose en verde si está vacío y en rojo en caso contrario.
- Cuando un objeto esté próximo al muro, indicar la proximidad mediante el buzzer, emitiendo un sonido con mayor frecuencia a medida que se acerca al muro y apagándose cuando se alcance una distancia ideal.
- Cuando un objeto desee ingresar, encender las luces LED para facilitar la visibilidad.
- Usar el servo para permitir o bloquear la entrada de objetos según la distancia medida.

III. Materiales y Equipos

- Computador
- Entorno de desarrollo integrado Thonny
- 1 buzzer
- 1 servo SG-90
- ESP32 Wroom
- Sensor ultrasónico HCSR04
- 5 LEDs
- 5 resistencias de 1kΩ
- Fuente DC de 5 voltios
- Cable de transferencia de datos
- Protoboard
- Jumpers

IV. Marco Teórico

Python

Python es un conjunto de herramientas versátil con una amplia gama de bibliotecas que se pueden usar para crear casi cualquier cosa, desde una calculadora sencilla hasta un sitio web complejo o un modelo de aprendizaje automático. Su diseño fácil de usar hace que sea accesible para cualquier persona que desee aprender y usar estas herramientas de manera efectiva.

Imagen 1: Logo de Python

MicroPython

Es una versión ligera y eficiente de Python 3 creada para funcionar en microcontroladores y sistemas embebidos. Su tamaño reducido le permite ejecutarse incluso en dispositivos con recursos muy limitados.

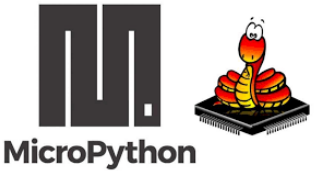

Imagen 2: Logo de MicroPython

ESP-32

El ESP32 es una familia de chips de bajo coste y consumo energético, que integra tecnología Wi-Fi y Bluetooth de modo dual.

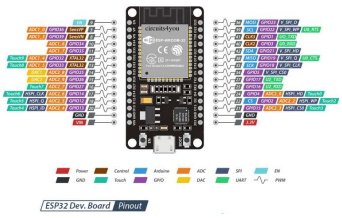

Imagen 3: ESP32

Sensor Ultrasónico

El sensor ultrasónico es un tipo de detector de proximidad que opera sin necesidad de contacto y puede identificar objetos ubicados desde pocos centímetros hasta varios metros de distancia.

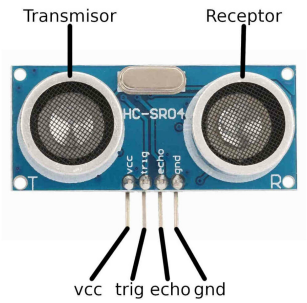

Imagen 4: Sensor Ultrasónico

Servo SG-90

El servo SG-90 es un actuador rotatorio compacto que permite controlar con precisión su posición angular. Puede girar entre 0° y 180° y opera con un voltaje de 4.8 a 6 VDC. Incluye tres brazos, tres tornillos y un cable de aproximadamente 25 cm de longitud.

Imagen 5: Servo SG-90

V. Desarrollo de la Práctica

Para el funcionamiento del sensor ultrasónico HC-SR04 se empleó una librería externa que facilita su uso, ya que incorpora una función destinada a calcular y mostrar en el monitor serial la distancia detectada en centímetros. Además, esta librería gestiona internamente el envío del pulso de disparo y la medición del tiempo de retorno del eco, simplificando la integración del sensor en el programa y garantizando lecturas más precisas y estables.

```
from machine import Pin, time_pulse_us
from time import sleep_us

class HC1SR04:
    def __init__(self, trigger_pin, echo_pin, echo_timeout_us=30000):
        self.trigger = Pin(trigger_pin, Pin.OUT)
        self.echo = Pin(echo_pin, Pin.IN)
        self.echo_timeout_us = echo_timeout_us

        # Asegurar trigger en bajo
        self.trigger.value(0)
        sleep_us(5)

    def distance_cm(self):
        # Enviar pulso de 10 microsegundos
        self.trigger.value(1)
        sleep_us(10)
        self.trigger.value(0)

        # Medir duración del pulso en echo
        duration = time_pulse_us(self.echo, 1, self.echo_timeout_us)

        if duration < 0:
            return -1  # error o fuera de rango

        # Convertir tiempo en distancia
        distance = (duration / 2) / 29.1
        return round(distance, 2)

    def distance_mm(self):
        d = self.distance_cm()
        if d < 0:
            return -1
        return int(d * 10)
```

Imagen 6: Código de funcionamiento del sensor ultrasónico

Para el control del servomotor se implementaron dos funciones: una función de mapeo que convierte el rango de ángulos (0° a 180°) en los valores de duty cycle requeridos por el PWM del servo, y una función de movimiento que aplica este valor al servomotor. Esta estructura permite posicionar el servo con precisión, simplifica el código principal y facilita la adaptación del sistema a diferentes servos o rangos de operación.

```
def mapx(x, in_min, in_max, out_min, out_max):
    return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

def mover_servo(angulo):
    servo1.duty(mapx(angulo, 0, 180, 20, 120))
```

Imagen 7: Control del servo

El sistema de control de LEDs o del buzzer se basa en la configuración de varias salidas digitales de la ESP32, cada una asociada a un LED que representa un estado específico del garaje. Mediante funciones simples como encender, apagar o hacer parpadear un LED, el programa puede comunicar visualmente condiciones importantes del sistema. Por ejemplo, el LED verde se activa cuando el garaje está libre, indicando

que no hay obstáculos; el LED rojo se enciende cuando se detecta la presencia de un vehículo, y el LED amarillo puede emplearse para alertas o advertencias mediante parpadeo. Esta implementación permite una señalización clara y directa, facilitando al usuario la interpretación del estado del garaje de manera inmediata y eficiente.

```
from machine import Pin
from time import sleep

led_rojo = Pin(12, Pin.OUT)      # LED indica vehículo presente
led_verde = Pin(14, Pin.OUT)     # LED indica garaje libre
led_amarillo = Pin(27, Pin.OUT)  # LED de advertencia (opcional)
def encender_led(led):
    led.value(1)

def apagar_led(led):
    led.value(0)

def parpadear_led(led, veces=3, tiempo=0.3):
    for _ in range(veces):
        led.value(1)
        sleep(tiempo)
        led.value(0)
        sleep(tiempo)

apagar_led(led_rojo)
encender_led(led_verde)

print("Simulando vehículo detectado...")
apagar_led(led_verde)
encender_led(led_rojo)
parpadear_led(led_amarillo, 5, 0.2)
```

Imagen 8: Salidas

El código de lectura del sensor, como se mencionó anteriormente, así como cada una de las respuestas según la distancia indicada, se encuentra dentro de un bucle ‘while True‘ para permitir que las lecturas se realicen sin condiciones en un bucle infinito.

Para indicar el estado encendido o apagado de las respuestas, se utiliza la función ‘.value()‘, donde el valor 0 indica apagado y 1 indica encendido. Para el servo, se usa la función ‘servo(servo1, 90)‘, donde en este caso el ‘servo1‘ se establece a 90 grados.

Mediante condicionales ‘if‘ y ‘else‘ se ejecutan todas las acciones según la medida proporcionada por el sensor ultrasónico.

```
# Reset LEDs y buzzer
led1.value(0)
led2.value(0)
led3.value(0)
ledVerde.value(0)
ledRojo.value(0)
buzzer.value(0)

# ---- LÓGICA ----
if distancia <= 4:
    ledRojo.value(1)
    buzzer.value(1)
    mover_servo(90)

elif distancia <= 10:
    ledRojo.value(1)
    buzzer.value(1)
    mover_servo(90)

elif distancia <= 20:
    led1.value(1)
    mover_servo(45)

else:
    ledVerde.value(1)
    mover_servo(0)

sleep(0.1)
```

Imagen 9: Condicionales para el control

Las distancias de referencia son entre 10 y 4 centímetros para encender el LED rojo que indica la presencia de un objeto, y fuera de esta distancia, el LED verde indica la disponibilidad. De igual manera, dentro de esta distancia, se enciende el buzzer, aumentando la frecuencia de su sonido conforme se acerca al sensor, y apagándose a los 4 cm para indicar que la posición de parqueo es ideal. Entre 25 y 5 cm, se encienden las luces que permiten la visibilidad (led1, led2, led3) y se cambia la posición inicial del servo de 0 a 90 grados, lo que permite el paso del objeto. Fuera de esta distancia, el servo regresa a su posición inicial de 0 grados.

VI. Conclusiones y Recomendaciones

- Cuando se utiliza la función ‘sleep‘ de la librería ‘time‘, es importante revisar el código, ya que esta función interrumpe la medición del sensor.
- Al dividir las acciones en condicionales, se facilita el trabajo, en comparación con el uso de bucles o la definición de funciones específicas.
- Las medidas del sensor no son completamente precisas, por lo que en ocasiones las respuestas esperadas según la distancia no son correctas.
- Implementar el sensor ultrasónico fue complejo debido a la falta de librerías nativas en el entorno de desarrollo. Fue necesario descargar una librería creada por terceros. De igual manera, aunque para el servo no se requiere una librería específica, también hubo problemas en su implementación.