servomotor-projekt

November 13, 2024

1 Servo-Motor Steuerung mit Raspberry Pi und Jupyter Notebook

1.1 1. Einführung

Dieses Jupyter Notebook führt in die praktische Anwendung der Steuerung von Servomotoren mit einem Raspberry Pi ein. Es kombiniert Elektronik und Python-Programmierung, um grundlegende Steuerungen für Servomotoren zu erstellen und zu testen. Das Projekt zeigt, wie man einen analogen Servomotor mit verschiedenen Bibliotheken steuert, die Grundlagen des PWM-Signale erklärt und den Aufbau der Schaltung beschreibt. Mit einer schrittweisen Anleitung und praxisnahen Experimenten bietet dieses Notebook eine solide Grundlage für Einsteiger in die Welt der Elektronik und die Nutzung des Raspberry Pi für Hardware-Projekte.

1.1.1 Projektziel

Das Ziel dieses Projekts ist es, einen Servomotor mit einem Raspberry Pi zu steuern und dabei die Prinzipien der Pulsweitenmodulation (PWM) anzuwenden.

1.1.2 Relevanz

Dieses Projekt ist relevant, da es die Grundlagen der Elektronik und Programmierung kombiniert und ein grundlegendes Verständnis für die Steuerung von Motoren, insbesondere Servomotoren, entwickelt.

Blink in Projektskomponenten:

Quellen: Servo Motor, Raspberry Pi 5

Führen Sie die folgenden zwei Zellen, um Videos zu Servo in einem neuen Browser-Tab zu öffnen.

[2]:
```
import webbrowser
url = "https://www.youtube.com/embed/4RGCWnhy888?start=4"
webbrowser.open_new_tab(url)
```

[2]:
```
True
```

[3]:
```
url = "https://www.youtube.com/embed/xHDT4CwjUQE?start=10"
webbrowser.open_new_tab(url)
```

[3]: True

1.2 2. Grundlagen und Theorie

1.2.1 Servomotoren

- **Funktionsweise:** Servomotoren sind Aktoren, die eine präzise Steuerung der Winkelposition ermöglichen. Sie bestehen aus einem Motor, einem Potentiometer und einer Steuerungselektronik.
- **Steuerung:** Servomotoren werden typischerweise mit einem PWM-Signal gesteuert, wobei die Pulsbreite die Position des Servos bestimmt.

Weiteres zu Servomotoren

1.2.2 PWM-Signale

- **Definition:** PWM steht für Pulsweitenmodulation, eine Technik zur Steuerung der Leistung in elektronischen Schaltungen, indem das Ein/Aus-Verhältnis eines Signals variiert wird.
- **Anwendung bei Servos:** Die Position des Servos wird durch die Breite des PWM-Signals gesteuert, wobei typische Servos Pulse zwischen 1 ms (Min-Position) und 2 ms (Max-Position) erwarten.

PWM-Typ	Beschreibung	PINs
Hardware-PWM	Verwendung von Hardware zur Generierung von PWM-Signalen	PWM0: GPIO 12 (Pin 32), GPIO 18 (Pin 12) PWM1: GPIO 13 (Pin 33), GPIO 19 (Pin 35)
Software-PWM	Nutzung von Software zur Erzeugung von PWM-Signalen	GPIO 18: Häufig genutzt, viele Tutorials GPIO 23: Beliebter Pin für SW-PWM GPIO 24: Gut geeignet für mehr PWM-Kanäle

Arten von PWM: Hardware-PWM (HW-PWM)
- Definition: HW-PWM wird durch spezielle Hardware-Register des Mikrocontrollers (z. B. Raspberry Pi) gesteuert.
- Vorteile: Bietet eine präzise und stabile PWM-Signalausgabe. Unabhängig von anderen Prozessen, die auf dem Gerät laufen.
- Verwendung: Ideal für zeitkritische Anwendungen wie Motorsteuerung, bei denen genaue Pulsweiten erforderlich sind.

Software-PWM (SW-PWM)
- Definition: SW-PWM wird in der Software implementiert, indem Pins durch Programmierung mit sleep() ein- und ausgeschaltet werden.
- Nachteile: Geringere Präzision und Stabilität; kann durch andere laufende Prozesse beeinflusst werden.
- Verwendung: Nützlich für einfache Anwendungen oder wenn Hardware-PWM nicht verfügbar ist.

Weiteres zu PWM-Signals

1.3 3. Materialien und Werkzeuge

1.3.1 Software und Hardware

Kategorie	Komponenten
Software	Raspbian OS
	Python 3
	Jupyter Notebook
Hardware	Raspberry Pi 5 Model B Rev 1.0
	Breadboard
	Jumper-Kabel
	GRV Servo Motor
	5V Stromversorgung für den Servo

1.3.2 Datenblätter

Servo-Datenblatt:
- Drehmoment: 21.0/25.2oz.in - Geschwindigkeit: 0.12/10.1 s/60° - Spannung: 4.8-6V

Links zum Datenblatt Sowie Servo Motor:
- Servo Datenblatt - Gravo Servo

1.3.3 Verdrahtung

Verbindung	Raspberry Pi Pin	Servo-Anschluss
Steuerungs-Pin des Servos	GPIO 18 (PWM)	Steuerungs-Pin (Signal)
Stromversorgung (Vcc)	-	Externe 5V-Quelle
Masse (GND)	GND	GND

1.3.4 PWM: GPIO-Pins

PWM Kanal	GPIO Pin	Pin Nummer
PWM1	GPIO 12	Pin No 36
PWM2	GPIO 16	Pin No 33
PWM1	GPIO 18	Pin No 12
PWM2	GPIO 19	Pin No 35

Gucken Sie die komplette Erklärung von Pins hier

Übung In diesen Beispielen wurde der GPIO18 genutzt. Bauen Sie die Schaltung um, um PWM2 (GPIO13) zu nutzen. Zeigen Sie dem Dozent Ihre neue Schaltung und testen Sie es.

Tipp!

Gucken Sie sich das Pinout-Bild.

1.4 4. Schaltungsdesign

1.4.1 Raspberry P i5 GPIO-Pinout

Das folgende Bild zeigt die PINs und Protokollen eines Rasüpberry Pi und die wichtigsten Funktionen, die mit PINs verbunden sind.

1.4.2 Zu Servo Motor

Dieses Bild stellt die PINs des Servos sowie einige Spezifikationen dar.

1.4.3 Schaltplan

So könnte die Schaltung eines Servos mit Rasp Pi ausshen.

Erstellt durch circuito.io

Alternativ
Dieses Bild druckt die PINs-Schaltung deutlisch klar aus.

Erstellt durch easyeda

1.4.4 Geschwindigkeit messen

Servo Motor kann mit einem Zeiger als Geschwindigkeit-Messer genutzt werden.
Im folgenden Bild ist die Anzeige der Geschwindigkeit genauso groß wie der Bewegungsgrad des Servo Motors, nämlich 180.

1.4.5 Komponentenauswahl

Komponente	Beschreibung
Servomotor	GRV Servo 21.0/25.2oz.in 0.12/10.1 s/60°
Jumper-Kabel	Für die Verbindung der GPIO-Pins mit dem Servo
Breadboard	Für die Schaltungsaufbauten

Für Weiteres über die Verwendung von Servo mit Rasp Pi klicken Sie hier.

1.5 5. Implementierung

1.5.1 Hardware-Aufbau

1. Komponenten auf dem Breadboard wie oben beschrieben platzieren und verkabeln.
2. Raspberry Pi aufstellen und durch USB-Kabel an Rechner mit Strom versorgen.

1.5.2 Software-Setup

1. Raspbian OS installieren.
2. Python und Jupyter Notebook installieren.

1.5.3 Installiere die RPi.GPIO und gpiozero Bibliotheken:

Für die Installationsanweisungen dieser Bibliotheken Klicken Sie hier

[ ]:
```
from gpiozero import Servo
from time import sleep

servo = Servo(18) #GPIO 18

try:
    while True:
        servo.mid()   # Mittelposition
        sleep(2)
        servo.min()   # Minimum Position
        sleep(2)
        servo.max()   # Maximum Position
        sleep(2)
except KeyboardInterrupt:
    pass
```

1.5.4 Funktionen für den Servomotor

Funktion	Beschreibung
`servo.mid()`	Stellt den Servomotor in die mittlere Position, also auf den mittleren Punkt seines Bewegungsbereichs.
`servo.min()`	Bewegt den Servomotor in die minimale Position, also den unteren Extrempunkt seines Bewegungsbereichs.
`servo.max()`	Bewegt den Servomotor in die maximale Position, also den oberen Extrempunkt seines Bewegungsbereichs.

Für weiteres über Servo-Bibliotheken und Funktionen klicken Sie hier.

Diese Impelimentierung lässt der Zeiger dies Servos auf drei Positionen bewegen, nämlich Mittelposition, Minimum Position und Maximum Position. Auch hier wurde der GPIO 18 benutzt.

Übung

- Mid, Min und Max wurden oben behandelt. Welche weitere Funktionen bietet uns Servo?
- Schreiben Sie diesen Beispiel nochmal, aber dieses Mal mit der Nutzung von RPi.GPIO Bibliothek.
- Anschließend testen Sie Ihren Code mit Rasp Pi. Funktioniert es noch wie vorhin? Zeigen Sie dem Dozent Ihre Änderungen.
- Welcher Wert (in [ms]) für die mittlere Position steht?

Lösung!

- servo.detach(): Dies deaktiviert den Servo und spart Energie, wenn er nicht benutzt wird.

- servo.value: Ein Attribut, das den aktuellen Wert des Servos (zwischen -1 und 1) zurückgibt, wo:

  -1 für die minimale Position,
  0 für die mittlere Position und
  1 für die maximale Position steht.

Für die mittlere Position eines Standard-Servomotors liegt der Puls bei ca. 1,5 ms. Dies entspricht typischerweise einem Duty Cycle von 7,5% bei einer 50 Hz PWM-Frequenz.

```
Min-Position: 1 ms (2% Duty Cycle)
Max-Position: 2 ms (12% Duty Cycle)
Mid-Position: 1,5 ms (7,5% Duty Cycle)
```

“‘bash import RPi.GPIO as GPIO from time import sleep

```
# Konfiguration der GPIO-Pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)  # Pin 18

# Servo initialisieren
pwm = GPIO.PWM(18, 50)  # 50 Hz
pwm.start(0)

# Funktion zur Steuerung des Servowinkels
def set_angle(angle):
    duty = angle / 18 + 2
    GPIO.output(18, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)  # Warte, bis der Servo sich bewegt hat
    GPIO.output(18, False)
    pwm.ChangeDutyCycle(0)

try:
    while True:
        angle = float(input("Geben Sie den gewünschten Winkel (0-180) ein: "))
        set_angle(angle)

except KeyboardInterrupt:
    pwm.stop()
    GPIO.cleanup()
```

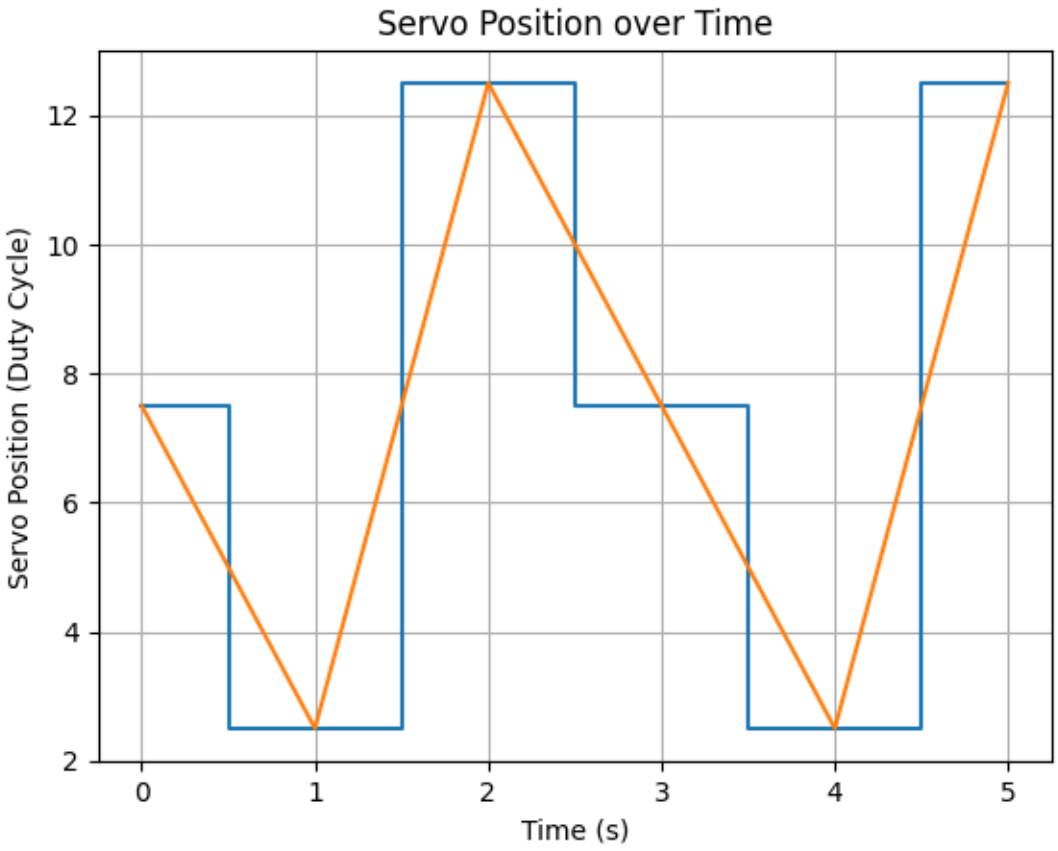
1.5.5 Darstellung in Matplotlib

Matplotlib kann verwendet werden, um die PWM-Signale oder die Bewegungsposition des Servos über die Zeit darzustellen.
Falls ie diese Bibliothek noch nicht installiert haben, Gucken Sie sich die Installation hier.

[8]:
```
import matplotlib.pyplot as plt
```

```
times = [0, 1, 2, 3, 4, 5]
positions = [7.5, 2.5, 12.5, 7.5, 2.5, 12.5]

plt.step(times, positions, where='mid')  # 'mid', 'pre', or 'post' kann␣
 ↪verwendet werden
plt.plot(times, positions)
plt.xlabel('Time (s)')
plt.ylabel('Servo Position (Duty Cycle)')
plt.title('Servo Position over Time')
plt.grid(True)
plt.show()
```



Die `step`-Funktion in Matplotlib ermöglicht es, die Linien zwischen den Datenpunkten entweder vertikal oder horizontal verlaufen zu lassen, anstatt diagonal. Sie können den Stil der Treppenlinien mit dem Parameter where steuern, wobei `mid`, `pre`, und `post` verschiedene Varianten darstellen.

1.5.6 Schritte in einem Diagramm

Schritt	Beschreibung
`pre`	Der Schritt in einem Diagramm beginnt direkt vor dem x-Wert.

Schritt	Beschreibung
`post`	Der Schritt beginnt direkt nach dem x-Wert.
`mid`	Der Schritt erfolgt in der Mitte zwischen zwei x-Werten.

Weiteres zu `step`- und Matplotlib-Funktionen klicken Sie hier.

1.5.7 Übung:

- Finden Sie heraus, welche andere Funktionen die Matplotlib anbietet?
- Im obigen Beispiel wurde der Servo zwischen drei Positionen bewegt.
  Erweitern Sie das Beispiel, um eine langsamere Bewegung zwischen den Positionen zu simulieren.

  Tipp!
  Nutzen Sie de Funktion sleep(x), wobei x die Schlafzeit ist.

1.6 6. Experimente und Ergebnisse

1.6.1 Versuchsaufbau

- Der oben beschriebene Hardware-Aufbau wurde verwendet.
- Der Code wurde in einem Jupyter Notebook ausgeführt.
- Der Servo bewegte sich entsprechend den vorgegebenen Positionen.

1.6.2 Beispiel 1: Schrittweise-Bewegung nach Angabe von Winkel

In diesem Beispiel wurde die Bibliotheken RPi.GPIO und time, die es schon mit Python installiert.

Sehen Sie sich auch The Top 5 Libraries for Every Developer

[ ]:
```
#Falls noch nicht installiert führen Sie diesen Befehl aus.
!pip3 install RPi.GPIO
```

[ ]:
```
#Bibliotheken importieren
import RPi.GPIO as GPIO
from time import sleep
```

[ ]:
```
# Konfiguration der GPIO-Pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)  # Pin 18
```

[ ]:
```
# Servo initialisieren
pwm = GPIO.PWM(18, 50)  # 50 Hz
pwm.start(0)
```

Da wir die GPIO.PWM-Klasse verwenden, nutzen wir Hardware-PWM (HW-PWM). Diese Methode ist stabiler und ermöglicht präzisere Steuerungen im Vergleich zur Software-PWM.

[ ]:
```
#Motor-Bewegung durch Eingabe von Winkel
def set_angle(angle):
    duty = angle / 18 + 2
    GPIO.output(18, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)  # Warte, bis der Servo sich bewegt hat
    GPIO.output(18, False)
    pwm.ChangeDutyCycle(0)
```

ChangeDutyCycle ist eine Methode, die in der Bibliothek RPi.GPIO verwendet wird, um die Duty-Cycle-Eigenschaft eines PWM-Signals zu ändern. Die Duty-Cycle beschreibt das Verhältnis von Ein- zu Ausschaltzeiten eines PWM-Signals (Pulsweitenmodulation). Für weiteres zu `RPi.GPIO` klicken Sie hier

[ ]:
```
try:
    while True:
        try:
            # Benutzereingabe für den gewünschten Winkel
            angle = float(input("Geben Sie den gewünschten Winkel (0-180) ein␣
 ↪(oder 'exit' zum Beenden): "))

            # Eingabewert überprüfen
            if angle < 0 or angle > 180:
                print("Ungültiger Winkel. Bitte geben Sie einen Wert zwischen 0␣
 ↪und 180 ein.")
                continue

            # Servo auf den angegebenen Winkel bewegen
            set_angle(angle)

        except ValueError:
            user_input = input("Ungültige Eingabe. Geben Sie 'exit' zum Beenden␣
 ↪ein: ")
            if user_input.lower()== 'exit':
                break
            else:
                print("Ungültige Eingabe. Bitte geben Sie einen Wert zwischen 0␣
 ↪und 180 ein.")

except KeyboardInterrupt:
    print("Programm wurde unterbrochen.")

finally:
    pwm.stop()
    GPIO.cleanup()
```

In diesem Beispiel wird man bei jedem Schritt danach gefordert, durch die Tastatur einen Winkel einzugeben. Der Servo bewegt sich dann aus der aktuellen Posiotion auf die neue durch den Winkel

angegebene Posiotion. Falls der Nutzer einen Wert größer 180, was der größtmögliche Winkel des Motors ist, oder kleiner 0, was das Minimum des Servo ist, kriegt man eine Ausgabe auf Jupyter, dass eine ungültige Eingabe erfolgt wurde. Motor macht dann nichts und wartet auf eine neue Eingabe.

Übung - Welcher Wert (in [ms]) steht für einen Winkel von 80 Grad?

Lösung! Der Wert für 80 Grad kann folgendermaßen berechnet werden:

Formel für den Duty Cycle: Duty=(angle/18)+2 Duty=18angle+2

Für 80 Grad: Duty=(80/18)+2=6.44 ms

1.6.3 Beispiel 2: dauerhafte Bewegung

Wir nutzen die gleiche Bibliotheken und PIN-Einstellungen wie oben. Merken Sie sich hierbei den Unterschied zwischen dem obigen und dem folgenden Code.

[ ]:
```
if __name__ == '__main__':
    degree = 180 #Maximum von Servo
    direction = 0 #Richtung
    try:
        while True: #Das heißt: dauerhafte Bewegung
            set_angle(degree) #Aufrufen der obigen Funktion im Beispiel 1
            print(degree) #Ausgabe, für Klarheit
            # Einstellung von Bewegungsrichtung anhand des Winkels.
            if degree == 180:
                direction = 1
            elif degree == 0:
                direction = 0

            if direction == 0:
                degree += 10
            else:
                degree -= 10
    except KeyboardInterrupt:
        pwm.stop()
        GPIO.cleanup()
```

In diesem Beispiel bewegt sich der Zeiger bzw. dreht sich der Motor ständig ohne anzuhalten. Der Servo kann nur zwischen Winkel 0 und 180. In diesem Fall bewegt sich er von 0 auf 180. Wenn der Servo 180 erreicht, bewegt sich zurück auf 0 usw. . Im Beispiel kann man bemerken, dass die Variable degree sich immer um 10 erhöht oder verkleinert. Dies ist die Größe eines Schritts des Servos. 10 wurde mit Absicht eingegeben, damit sich der Servo während Experimenten nicht sehr langsam bewegt.

1.6.4 Übung:

- Suchen Sie sich einen anderen Pin, wo Sie den Servo mit einem anderen PWM-Pin verbinden und passen Sie den Code an. Funktioniert der Servo immer noch?

- Tipp!

  Gucken Sie sich das Pinout-Bild.

- Ändern Sie den Code so, dass der Servo nur 4 Schritte zwischen 0 und 180 machen kann. Wie machen Sie das?

- Lösung!

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

pwm = GPIO.PWM(18, 50)  # 50 Hz
pwm.start(0)

def set_angle(angle):
    duty = angle / 18 + 2
    GPIO.output(18, True)
    pwm.ChangeDutyCycle(duty)
    sleep(1)
    GPIO.output(18, False)
    pwm.ChangeDutyCycle(0)

try:
    steps = [0, 60, 120, 180]
    while True:
        for angle in steps:
            print(f"Bewege Servo auf {angle} Grad")
            set_angle(angle)
            sleep(2)

except KeyboardInterrupt:
    pwm.stop()
    GPIO.cleanup()
```

“‘

1.6.5 Zusätzliche Ressourcen

Für besseres Verstand und für die Klarheit gucken Sie sich die folgenden Notebooks:
- Grundlagen-Notebook - Bib-Installationen - Raspbian OS

Sowie die Webseiten: - jupyter.org - mybinder