

Complete Verbatim Chat: Servo Motor Control

User: I'm considering changing the blast gate mechanism for my automated dust collection system. I might use a 'pivot gate' operated by a small servo motor that pivots the wedge-shaped gate through 90 degrees around one point instead of a linear gate controlled by a linear actuator that slides the gate in and out. Therefore, I need to learn something about how small servo motors are controlled and how to use a Raspberry Pi to control them. The angle of rotation is on the order of 90 degrees or so, the mass is on the order of a pound or less, speed should be less than 2 seconds for 90 degrees of rotation, cost should be fairly low (around \$10 or less per servo) and I've got both 5 volt and 12-volt supplies to run it. Please explain servo motors for small embedded systems in a straightforward and useful way for someone who has experience in software engineering (but not hardware) in the robotics field.

Assistant: Servo motors are a great choice for your application, as they allow precise control of angular position, speed, and torque within a compact form factor. Here's an overview of servo motors and how to control them using a Raspberry Pi:

****What is a Servo Motor?****

A servo motor is a small device with a control circuit, a motor, and a gearbox in one package. It's commonly used for precise angular positioning. Key features include:

- ****Control****: The position is controlled by a PWM (Pulse Width Modulation) signal.
- ****Range****: Most hobby servo motors rotate 0 - 180 degrees, though some offer full 360-degree continuous rotation.
- ****Torque****: Suitable for light loads like your application (~1 pound or less).

****How a Servo Motor Works****

- ****PWM Control****: The motor's angular position is determined by the width of the PWM pulse:
 - A typical servo expects a signal repeating every 20 ms (50 Hz).
 - The pulse width determines the position:
 - ~1 ms pulse = 0 degrees.
 - ~1.5 ms pulse = 90 degrees (neutral).
 - ~2 ms pulse = 180 degrees.
- ****Power****: Servos usually operate at 5 - 7.4V. A higher voltage increases torque.

****Selecting a Servo Motor****

Given your requirements:

1. ****Torque****: Calculate based on the load's weight and the gate's pivot distance. A torque rating of 4 - 5 kgcm (~50 - 70 ozin) should suffice.
2. ****Speed****: Most servos complete 60 degrees in 0.1 - 0.2 seconds at their rated voltage.
3. ****Budget****: Servos like the MG995 or MG996R (~\$10 each) are affordable, provide high torque, and are readily available.

****Using a Raspberry Pi to Control a Servo****

1. ****Hardware Setup****:

- Connect the servo's control wire to one of the Raspberry Pi's GPIO pins.
- Provide power through an external 5 - 6V supply, ensuring the ground is shared between the Pi and the servo power source.

2. ****Software Setup****:

- Use Python with a library like ****RPi.GPIO**** or ****pigpio**** for PWM signal generation.

3. ****Basic Code Example**** (using RPi.GPIO):

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

```

Pin setup
GPIO.setmode(GPIO.BCM)
servo_pin = 18 # GPIO pin connected to the servo
GPIO.setup(servo_pin, GPIO.OUT)

PWM setup
pwm = GPIO.PWM(servo_pin, 50) # 50Hz frequency
pwm.start(0) # Initial duty cycle of 0

def set_angle(angle):
 duty = 2 + (angle / 18) # Map angle (0-180) to duty cycle (2-12)
 GPIO.output(servo_pin, True)
 pwm.ChangeDutyCycle(duty)
 sleep(0.5) # Allow the servo to reach position
 GPIO.output(servo_pin, False)
 pwm.ChangeDutyCycle(0)

try:
 set_angle(90) # Move to 90 degrees
 sleep(1)
 set_angle(0) # Move back to 0 degrees
finally:
 pwm.stop()
 GPIO.cleanup()
...

```

#### **\*\*Advantages of Pivot Gates with Servos\*\***

- Simplicity: No need for linear actuators or complex linkages.
- Precision: Control over exact gate position.
- Compactness: Takes up less space.

Would you like guidance on any specific part of this setup?

User: OK -- thanks. How do I reverse the direction of the motor to bring the gate back to its original position?

Assistant: Servo motors inherently allow bidirectional movement, meaning they can rotate back to their original position simply by commanding them to a lower angle. You don't need to reverse the motor's electrical connections (as you would with DC motors). Instead, you just specify the desired target angle using the same PWM control.

#### **\*\*Returning the Gate to Its Original Position\*\***

For example:

1. To open the gate, you command the servo to a specific angle, e.g., 90.
2. To close the gate, you command the servo back to 0.

Here's how this works in code (continuing from the earlier example):

```

```python
# Open the gate by moving to 90 degrees
set_angle(90)
sleep(1) # Keep it open for a while

# Close the gate by moving back to 0 degrees
set_angle(0)
...

```

****Why This Works****

- The servo interprets the PWM signal and adjusts its position accordingly.
- When you specify an angle lower than its current position (e.g., going from 90 to 0), the motor rotates in the reverse direction automatically.

****Important Notes****

- Ensure the physical mechanism allows free movement in both directions. If there's resistance (e.g., dust buildup), it could stress the servo.
- Always allow the servo enough time to reach its target position before issuing another command. This prevents the servo from stalling.

Would you like to implement a complete loop for testing this movement with delays, or do you have additional constraints?

User: Thanks, but not yet. I have a question: is the amount of travel relative to the current position (say, halfway) or is it the case that when at the midpoint it is told to go to the midpoint, it goes to the end?

Assistant: Servo motors operate using absolute positioning, not relative positioning. This means:

- When you command the servo to a specific angle (e.g., 45, 90, or 180), it moves to that exact position, regardless of its current position.
- The movement is not relative to its current position. If the servo is currently at 90 and you send a command to move to 45, it will move directly to 45, not 135.

****Example Behavior****

1. If the servo is at 90:

- Command it to go to 0 it moves to 0.
- Command it to go to 180 it moves to 180.

2. If the servo is at 45:

- Command it to go to 90 it moves to 90.
- Command it to go to 45 again it stays at 45 (no movement).

****How to Use Relative Positioning****

If you need ****relative positioning**** (e.g., "move 45 from where you are"), you can calculate the desired position manually in your code:

```
```python
Assume current_position keeps track of the servo's angle
current_position = 90 # Start at 90 degrees

Move +45 degrees relative to current position
new_position = current_position + 45
set_angle(new_position)

Update the current position
current_position = new_position
```
```

In this approach:

- You maintain a variable (`current_position`) that tracks where the servo is.
- Use this variable to compute relative movements by adding or subtracting from it.

If this addresses your question, let me know. If you'd like help with implementing relative positioning or managing current positions in your setup, I can assist!

User: No, thanks - absolute positioning is just what I wanted!

Assistant: You're welcome - absolute positioning simplifies things significantly! It's great for ensuring the gate moves predictably to predefined positions. If you need more assistance with implementing or testing your servo setup, feel free to reach out. Happy building!