

## Calling an API to Control the Robot

### Calling an API to Control the Robot

- The reason why the movement and other functions of walking robots are much more complicated than wheeled robots:

1. For wheeled robots, you only need to control the high and low levels of a few GPIO ports to control their movement.
2. For walking robots, you need to control the angle change of a dozen servos, which involves the movement direction and gait generation, and write a lot of code to realize their movements.

- If the code related to the motion control of the walking robot is written into ``server.py`` or ``webServer.py``, the program will be less readable, so we separate the communication part from the underlying motion control program.

- The code for controlling the movement of the robot is in ``SpiderG.py`` in the ``server`` folder.

- If you want to change the communication method of the robot or write a program to control the robot without reconstructing the code of the underlying motion control and gait generation, you may refer to the API introduced in this chapter. This chapter also introduces the method used by our ``webServer.py`` or ``server.py`` to control the robot movement.

- First of all, your program needs to be placed in the same file directory with `SpiderG.py`.  
Importing `SpiderG.py`:

```
import SpiderG
```

- All servos are set to the initial position, and the default value is 300. This value will be changed after fine-tuning of the servo position.

```
SpiderG.move_init()
```

- Control the robot to move forward, yet it will not hinder the movement controlled by the main program. Similar for the next steps:

```
SpiderG.walk('forward')
```

- Control the robot to move forward for ten seconds and then stop:

```
import SpiderG
import time

SpiderG.walk('forward')
time.sleep(10)
SpiderG.servoStop()
```

- Control the robot to turn left:

```
SpiderG.walk('turnleft')
```

- Control the robot to turn right:

```
SpiderG.walk('turnright')
```

- Control the robot to turn on self-balancing mode:

```
SpiderG.steadyModeOn()
```

- Control the robot to turn on the left-tilt mode:

```
SpiderG.walk('Lean-L')
```

- Control the robot to turn on the right-tilt mode:

```
SpiderG.walk('Lean-R')
```

- You can also use more primitive and basic programs to control the tilt of the robot body:

```
SpiderG.status_GenOut(height_input, pitch_input, roll_input)  
SpiderG.direct_M_move()
```

**height\_input:** Setting the height change of the robot, ranging from -200 to 200. The smaller the value, the higher the robot.

**pitch\_input:** Setting the height change of the robot head, ranging from -200 to 200. When the value is -200, the robot raises its head; when it is 200, the robot lowers its head.

**roll\_input:** Setting the robot's left and right tilt angle, ranging from -200 to 200. When the value is -200, the robot tilts to the right; when it is 200, the robot tilts to the left.

The **status\_GenOut()** function is used to set the target position, but the robot will not move to this position.

The **direct\_M\_move()** function is used to adjust the robot servo to a new state.

● You can change the state adjustment function according to your needs by changing the `status_GenOut(height_input, pitch_input, roll_input)` of *SpiderG.py*. Below is the source code for it:

```

1. def ctrl_range(raw, max_genout, min_genout):
2.     """
3.     This function is used to limit the PWM value of the servo to a certain range.
4.     """
5.     if raw > max_genout:
6.         raw_output = max_genout
7.     elif raw < min_genout:
8.         raw_output = min_genout
9.     else:
10.        raw_output = raw
11.    return int(raw_output)
12.
13. def status_GenOut(height_input, pitch_input, roll_input):
14.     """
15.     Calculating the deviation of the middle joint of each leg according to pitch_input and roll_input.
16.     """
17.    FL_input = wiggle_v*pitch_input + wiggle_v*roll_input
18.    FR_input = wiggle_v*pitch_input - wiggle_v*roll_input
19.
20.    HL_input = - wiggle_v*pitch_input + wiggle_v*roll_input
21.    HR_input = - wiggle_v*pitch_input - wiggle_v*roll_input
22.
23.    """
24.    Based on the reference initial value of each servo, the changed PWM value is calculated and output to goal_dict to save.
25.    """
26.    def leg_FL_status():
27.        goal_dict['FLB'] = FLB_init_pwm
28.        goal_dict['FLM'] = ctrl_range(int(FLM_init_pwm + (height_input + FL_input)*FLM_direction),
29.                                     max_dict['FLM'], min_dict['FLM'])
30.        goal_dict['FLE'] = FLE_init_pwm
31.
32.    def leg_FR_status():
33.        goal_dict['FRB'] = FRB_init_pwm
34.        goal_dict['FRM'] = ctrl_range(int(FRM_init_pwm + (height_input + FR_input)*FRM_direction),
35.                                     max_dict['FRM'], min_dict['FRM'])
36.        goal_dict['FRE'] = FRE_init_pwm
37.

```

```

38. def leg_HL_status():
39.     goal_dict['HLB'] = HLB_init_pwm
40.     goal_dict['HLM'] = ctrl_range(int(HLM_init_pwm + (height_input + HL_input)*HLM_direction),
41.                                   max_dict['FRM'], min_dict['FRM'])
42.     goal_dict['HLE'] = HLE_init_pwm
43.
44. def leg_HR_status():
45.     goal_dict['HRB'] = HRB_init_pwm
46.     goal_dict['HRM'] = ctrl_range(int(HRM_init_pwm + (height_input + HR_input)*HRM_direction),
47.                                   max_dict['FRM'], min_dict['FRM'])
48.     goal_dict['HRE'] = HRE_init_pwm
49.
50. leg_FL_status()
51. leg_FR_status()
52. leg_HL_status()
53. leg_HR_status()
54. print(goal_dict['FLM'])

```

- Some variables related to the robot's walking in *SpiderG.py*:

```

1. '''
2. Defining the servo port number of the corresponding joint.
3. F stands for front legs.
4. H stands for hind legs.
5. L stands for the left leg.
6. R stands for the right leg.
7. B stands for the servo that controls the leg to swing back and forth (the servo fixed on the robot body).
8. M stands for the servo that controls the leg to swing up and down (the servo above and below the robot leg).
9. E stands for the servo that controls the the leg to swing far and near (the servo above and below the robot leg).
10. '''
11. FLB_port = 0
12. FLM_port = 1
13. FLE_port = 2
14.
15. FRB_port = 6
16. FRM_port = 7
17. FRE_port = 8
18.
19. HLB_port = 3

```

```
20. HLM_port = 4
21. HLE_port = 5
22.
23. HRB_port = 9
24. HRM_port = 10
25. HRE_port = 11
26. '''
27. Defining the PWM value of the initial position of each servo.
28. Servo fine-tuning related functions will change these values.
29. Initial value is 300.
30. '''
31. FLB_init_pwm = 300
32. FLM_init_pwm = 300
33. FLE_init_pwm = 300
34.
35. FRB_init_pwm = 300
36. FRM_init_pwm = 300
37. FRE_init_pwm = 300
38.
39. HLB_init_pwm = 300
40. HLM_init_pwm = 300
41. HLE_init_pwm = 300
42.
43. HRB_init_pwm = 300
44. HRM_init_pwm = 300
45. HRE_init_pwm = 300
46. '''
47. Defining the swing direction of the servo, which can be 1 or -1.
48. This value usually keeps the default. If a servo rotates inversely due to different factory batches,
49. you can change the corresponding movement direction parameters.
50. '''
51. FLB_direction = 1
52. FLM_direction = -1
53. FLE_direction = -1
54.
55. FRB_direction = -1
56. FRM_direction = 1
57. FRE_direction = 1
58.
59. HLB_direction = -1
60. HLM_direction = 1
61. HLE_direction = 1
```

```
62.
63. HRB_direction = 1
64. HRM_direction = -1
65. HRE_direction = -1
66. '''
67. Defining the lifting height of the swinging pair.
68. '''
69. wiggle_h = 120
70.
71. '''
72. Defining stride size.
73. '''
74. wiggle_v = 200
75.
76. '''
77. Defining the deviation of the midpoint of the support pair.
78. '''
79. wiggle_middle = 30
80.
81. '''
82. Defining the delay time of each cycle. The robot has to deal with other tasks in addition to moving,
83. so the delay time should not be 0.
84. Even if it is 0, the robot will not move fast due to I2C communication delay.
85. '''
86. deley_time = 0.02
87.
88. '''
89. Interpolation number between adjacent positions. The larger the value, the smoother the robot's movement.
90. '''
91. total_count = 3
```