

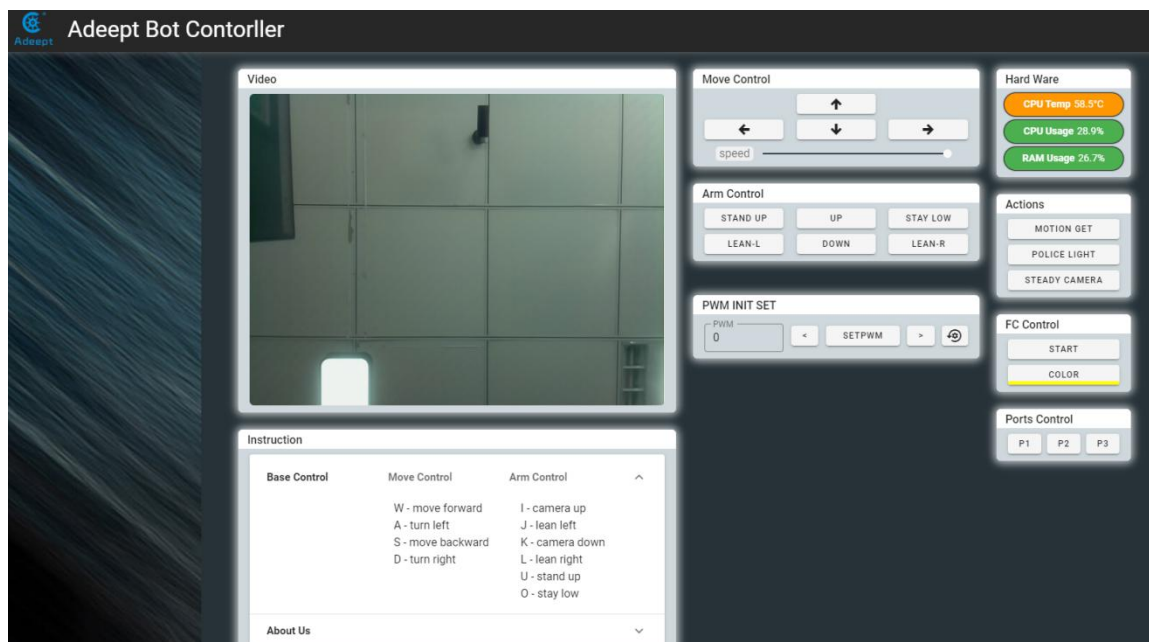
## Lesson 3 Introducing the Self-Balancing Function

### 2.1 Function Overview

The self-balancing function is developed based on MPU6050. After the function is enabled, you can place the robot on a panel. Slowly tilt the panel and you can see the robot adjust the height of its legs to keep balanced. When the function's working, the robot can't take any other actions.

### 2.2 Running the Self-Balancing Function

1. Start the DarkPaw Robot. It may take about 30-50s to boot.
2. After DarkPaw is turned on, open the Chrome browser on your mobile or computer, enter the IP address of your Raspberry Pi and access port ":5000" into the IP address bar, like this: [192.168.3.44:5000](http://192.168.3.44:5000). The web controller will then be displayed on the browser.



3. Click "[STEADY CAMERA](#)", and DarkPaw will keep a balance itself.

4. Click the button again to stop the function.

## 2.3 Main Program

The code of this tutorial lies in the [SpiderG.py](#). Here only the structure of the program is mentioned.

The code run based on some function or parameters in the [SpiderG.py](#).

```

1. def ctrl_range(raw, max_genout, min_genout):
2.     """
3.     This function is used to limit the servo PWM 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.    Pose control function has been introduced in the previous chapter.
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. def leg_FL_status():
23.    goal_dict['FLB'] = FLB_init_pwm
24.    goal_dict['FLM'] = ctrl_range(int(FLM_init_pwm + (height_input + FL_input)*FLM_direction),
25.                                  max_dict['FLM'], min_dict['FLM'])
26.    goal_dict['FLE'] = FLE_init_pwm
27.
28. def leg_FR_status():
29.    goal_dict['FRB'] = FRB_init_pwm
30.    goal_dict['FRM'] = ctrl_range(int(FRM_init_pwm + (height_input + FR_input)*FRM_direction),
31.                                  max_dict['FRM'], min_dict['FRM'])
32.    goal_dict['FRE'] = FRE_init_pwm
33.
34. def leg_HL_status():

```

```

35. goal_dict['HLB'] = HLB_init_pwm
36. goal_dict['HLM'] = ctrl_range(int(HLM_init_pwm + (height_input + HL_input)*HLM_direction),
37.                               max_dict['FRM'], min_dict['FRM'])
38. goal_dict['HLE'] = HLE_init_pwm
39.
40. def leg_HR_status():
41. goal_dict['HRB'] = HRB_init_pwm
42. goal_dict['HRM'] = ctrl_range(int(HRM_init_pwm + (height_input + HR_input)*HRM_direction),
43.                               max_dict['FRM'], min_dict['FRM'])
44. goal_dict['HRE'] = HRE_init_pwm
45.
46. leg_FL_status()
47. leg_FR_status()
48. leg_HL_status()
49. leg_HR_status()
50. print(goal_dict['FLM'])
51.
52. def steady():
53.     global sensor
54.     """
55.     Determining whether the self-stabilization function is turned on.
56.     """
57.     if steadyMode:
58.         """
59.         Determining whether the MPU6050 is connected.
60.         """
61.         if MPU_connection:
62.             try:
63.                 """
64.                 Reading the data of MPU6050.
65.                 """
66.                 accelerometer_data = sensor.get_accel_data()
67.                 """
68.                 The data read by the Kalman filter algorithm.
69.                 """
70.                 X = accelerometer_data['x']
71.                 X = kalman_filter_X.kalman(X)
72.                 Y = accelerometer_data['y']
73.                 Y = kalman_filter_Y.kalman(Y)
74.                 """
75.                 Calculating angle deviation.
76.                 """

```

```

77.         X_error = X-X_steady
78.         Y_error = Y-Y_steady
79.         """
80.         mpu_tor is the action threshold of the self-stabilization mode. If the deviation exceeds this threshold, the robot will start to move. If the deviation is less than this threshold, the robot will not move. Since the deviation is inevitable, in order to avoid the robot constantly shaking, we use this threshold to judge whether angle compensation is required.
81.         """
82.         if abs(X_error)>mpu_tor or abs(Y_error)>mpu_tor:
83.             """
84.             The error is applied to the pose control function to compensate for the deviation. The P value is the proportional integral in automatic control. The larger the value, the more sensitive the robot movements. However, excessive proportional integral will cause the robot to overshoot and cause severe jitter.
85.             """
86.             """
87.             status_GenOut(0, X_error*P, Y_error*P)
88.             """
89.             The calculated target posture
90.             """
91.             direct_M_move()
92.         except:
93.             time.sleep(0.1)
94.             """
95.             There is a certain probability that the MPU6050 will cause a connection failure when performing high-frequency reading. If the connection fails, then you can re-establish the I2C connection with the MPU6050.
96.             """
97.             sensor = mpu6050(0x68)

```