

Getting to Know Gait of the DarkPaw Robot

Introduce Robot Gaits

- For a quadruped robot, the gait generation method is almost the most complicated part of the program because it needs to coordinate dozens of servos to move at the same time. Keeping every moment walking forward and backward, there should be at least three stance legs, which means only one leg can be in the swing phase at any moment, and at least three legs should be in the support pair.
- In order to make it more intuitive, we represent the position of each leg of the robot as 1, 2, 3, 4, 5, 6, 7, 8 respectively. Among them, position 1 is the swing pair, and the rest are the support pairs. The number 2 and 8 represents the two endpoints of the support pair, and the number 3, 4, 5, 6, 7 are the interpolation between the two support pairs. The actual position spacing represented by adjacent numbers is the same.
- Taking forward heading of the robot as the forward direction, we name the left front leg 'I', the left hind leg 'II', the right front leg 'III', and the right hind leg 'IV'.
- In order to coordinate the gait of the four legs, we use a global gait parameter. This parameter is used to coordinate the position of the four legs at a certain moment, as shown in the following table.

Code\Global Gait	1	2	3	4	5	6	7	8
I	1	2	3	4	5	6	7	8
II	3	4	5	6	7	8	1	2
III	5	6	7	8	1	2	3	4
IV	7	8	1	2	3	4	5	6

- For example, when the global gait is 3, leg I is in the 3 position, leg II in the 5 position, leg III in the 7 position, and leg IV in the 1 position.
- When the global gait changes to a cycle of 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3..., the robot moves forward. When the global gait changes to a cycle of 8, 7, 6, 5, 4, 3, 2, 1, 8, 7 ..., the robot moves backward.
- When the robot turns left or right, we use the diagonal gait for steering movement, which is much simpler than the triangular gait.

Code\Global Gait	1	2	3	4
I	1	2	5	8
II	5	8	1	2
III	5	8	1	2
IV	1	2	5	8

- When the global gait changes to a cycle of 1, 2, 3, 4, 1, 2, ..., the robot moves forward in a diagonal gait.
- When the global gait changes to a cycle of 4, 3, 2, 1, 4, 3, ..., the robot walks backward in a diagonal gait.
- When the two legs on the right side of the robot move forward and the two on the left side do backwards, the robot turns to the left.
- When the left two legs walk forward and the two on the right do backwards, the robot turns to the right.

Program of Gait Generation

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 move_smooth_base(servo_name, goal_pwm, old_pwm, now_pos, total_range):
2.     ....
3.     This function is used to control the gentle movement of the servo. It is the underlying
       function of move_smooth_goal() and does not cause blocking.
4.     ...
5.     pwm_input = int(old_pwm+(goal_pwm-old_pwm)*now_pos/total_range)
6.     pwm.set_pwm(servo_name, 0, pwm_input)
7.     return pwm_input
8.
9.
10. def direct_M_move():
11.     ....
12.     This function is used to control the fast movement of the servo without blocking.
13.     ...
14.     pwm.set_pwm(FLM_port, 0, goal_dict['FLM'])
15.     pwm.set_pwm(FRM_port, 0, goal_dict['FRM'])
16.     pwm.set_pwm(HLM_port, 0, goal_dict['HLM'])
17.     pwm.set_pwm(HRM_port, 0, goal_dict['HRM'])
18.     old_dict['FLM'] = goal_dict['FLM']
19.     old_dict['FRM'] = goal_dict['FRM']
20.     old_dict['HLM'] = goal_dict['HLM']
21.     old_dict['HRM'] = goal_dict['HRM']
22.
23.
24. def move_smooth_goal():
25.     ....
26.     This function is used to control the servo to move slowly to the target point. The bot
       tom function is move_smooth_base().
27. This function will cause blocking, because the for loop is used to achieve the effect of sl
       ow movement.
28. For more basic knowledge of slow motion of the servo, please refer to Controlling Slow Moti
       on of the Servo.
29.
30.     ...
31.     global now_command
32.

```

```

33.     if gait_set == 0 or now_command == 'turnleft' or now_command == 'turnright':
34.         .....
35.         For the diagonal gait, since the global gait interpolation becomes less, the number
           of interpolations between the two positions of the servo is multiplied by 3.
36.
37.         ...
38.         count_input = total_count*3
39.     elif gait_set == 1:
40.         .....
41.         For triangular gait, the number of interpolations between the two points of the ser
           vo is unchanged.
42.         ...
43.         count_input = total_count
44.
45.     for i in range(0, count_input):
46.         .....
47.         count_input is the number of interpolations between the two points of the servo. The
           higher the value, the slower the robot movement speed.
48.         ...
49.         if goal_command != now_command:
50.             .....
51.             If the command changes, we exit the for loop to avoid blocking and reduce latency.
52.             ...
53.             update_old()
54.             now_command = goal_command
55.             return 1
56.
57.         .....
58.         The following code is used to generate the position where the corresponding servo s
           hould move.
59.         ...
60.         now_dict['FLB'] = move_smooth_base(FLB_port, goal_dict['FLB'], old_dict['FLB'], i,
           count_input)
61.         now_dict['FLM'] = move_smooth_base(FLM_port, goal_dict['FLM'], old_dict['FLM'], i,
           count_input)
62.         now_dict['FLE'] = move_smooth_base(FLE_port, goal_dict['FLE'], old_dict['FLE'], i,
           count_input)
63.
64.         now_dict['FRB'] = move_smooth_base(FRB_port, goal_dict['FRB'], old_dict['FRB'], i,
           count_input)
65.         now_dict['FRM'] = move_smooth_base(FRM_port, goal_dict['FRM'], old_dict['FRM'], i,

```

```

count_input)
66.     now_dict['FRE'] = move_smooth_base(FRE_port, goal_dict['FRE'], old_dict['FRE'], i,
count_input)
67.
68.     now_dict['HLB'] = move_smooth_base(HLB_port, goal_dict['HLB'], old_dict['HLB'], i,
count_input)
69.     now_dict['HLM'] = move_smooth_base(HLM_port, goal_dict['HLM'], old_dict['HLM'], i,
count_input)
70.     now_dict['HLE'] = move_smooth_base(HLE_port, goal_dict['HLE'], old_dict['HLE'], i,
count_input)
71.
72.     now_dict['HRB'] = move_smooth_base(HRB_port, goal_dict['HRB'], old_dict['HRB'], i,
count_input)
73.     now_dict['HRM'] = move_smooth_base(HRM_port, goal_dict['HRM'], old_dict['HRM'], i,
count_input)
74.     now_dict['HRE'] = move_smooth_base(HRE_port, goal_dict['HRE'], old_dict['HRE'], i,
count_input)
75.     ....
76.     Changing this delay time can change the robot's movement speed. The longer the dela
y time, the slower the movement speed, but it will not become smoother. For smoother, you n
eed to increase the total_count variable.
77.     ...
78.     time.sleep(deley_time)
79.
80.     ....
81.     Updating the final position of the servo as the initial point of the next gentle movemen
t.
82.     ...
83.     pwm.set_pwm(FLM_port, 0, goal_dict['FLM'])
84.     pwm.set_pwm(FRM_port, 0, goal_dict['FRM'])
85.     pwm.set_pwm(HLM_port, 0, goal_dict['HLM'])
86.     pwm.set_pwm(HRM_port, 0, goal_dict['HRM'])
87.     old_dict['FLB'] = goal_dict['FLB']
88.     old_dict['FLM'] = goal_dict['FLM']
89.     old_dict['FLE'] = goal_dict['FLE']
90.
91.     old_dict['FRB'] = goal_dict['FRB']
92.     old_dict['FRM'] = goal_dict['FRM']
93.     old_dict['FRE'] = goal_dict['FRE']
94.
95.     old_dict['HLB'] = goal_dict['HLB']
96.     old_dict['HLM'] = goal_dict['HLM']

```

```

97.     old_dict['HLE'] = goal_dict['HLE']
98.
99.     old_dict['HRB'] = goal_dict['HRB']
100.    old_dict['HRM'] = goal_dict['HRM']
101.    old_dict['HRE'] = goal_dict['HRE']
102.
103.    old_dict['P'] = goal_dict['P']
104.    old_dict['T'] = goal_dict['T']
105.    return 0
106.
107.
108. def goal_GenOut(position_input, left_direction, right_direction):
109.     """
110.         This method is used for gait generation of quadruped robots. It is the code of the ga
it table introduced above.
111. This function can generate triangle gait and diagonal gait.
112. It can control the direction of the left leg and the right leg. 1 is forward, and -1 is b
ackward.
113. """
114. def leg_FL(pos, direction_input):
115.     """
116.         Entering the pos parameter to generate the PWM value of the leg at that position,
117. it is the gait generation method of the left front leg.
118.         """
119.         if pos == 1:
120.             goal_dict['FLB'] = int(FLB_init_pwm + (wiggle_middle)*FLB_direction)
121.             goal_dict['FLM'] = int(FLM_init_pwm + (wiggle_v - FL_height)*FLM_direction)
122.             goal_dict['FLE'] = int(FLE_init_pwm + (wiggle_v + 0)*FLE_direction)
123.         elif pos == 2:
124.             goal_dict['FLB'] = int(FLB_init_pwm + (wiggle_middle + wiggle_h*direction_inpu
t)*FLB_direction)
125.             goal_dict['FLM'] = int(FLM_init_pwm - FL_height*FLM_direction)
126.             goal_dict['FLE'] = int(FLE_init_pwm)
127.         else:
128.             goal_dict['FLB'] = int(FLB_init_pwm + (wiggle_middle + (wiggle_h*(6-(pos-2))/
3 -
129.                                     wiggle_h)*direction_i
nput)*FLB_direction)
130.             goal_dict['FLM'] = int(FLM_init_pwm - FL_height*FLM_direction)
131.             goal_dict['FLE'] = int(FLE_init_pwm)

```

```

132.         #print('FL: %d'%pos)
133. def leg_FR(pos, direction_input):
134.     .....
135.     Entering the pos parameter to generate the PWM value of the leg at that position, i
        t id the gait generation method of the right front leg.
136.     ...
137.     if pos == 1:
138.         goal_dict['FRB'] = int(FRB_init_pwm + (wiggle_middle)*FRB_direction)
139.         goal_dict['FRM'] = int(FRM_init_pwm + (wiggle_v - FR_height)*FRM_direction)

140.         goal_dict['FRE'] = int(FRE_init_pwm + (wiggle_v + 0)*FRE_direction)
141.     elif pos == 2:
142.
143.         goal_dict['FRB'] = int(FRB_init_pwm + (wiggle_middle + wiggle_h*direction_inp
        ut)*FRB_direction)
144.         goal_dict['FRM'] = int(FRM_init_pwm - FR_height*FRM_direction)
145.         goal_dict['FRE'] = int(FRE_init_pwm)
146.     else:
147. goal_dict['FRB'] = int(FRB_init_pwm + (wiggle_middle + (wiggle_h*(6-(pos-2)))/3 -
148.                                     wiggle_h)*direction_i
        nput)*FRB_direction)
149.         goal_dict['FRM'] = int(FRM_init_pwm - FR_height*FRM_direction)
150.         goal_dict['FRE'] = int(FRE_init_pwm)
151.         #print('FR: %d'%pos)
152. def leg_HL(pos, direction_input):
153.     .....
154.     Entering the pos parameter to generate the PWM value of the leg at that position,
        it is the gait generation method of the right front leg.
155.     ...
156.     if pos == 1:
157.         goal_dict['HLB'] = int(HLB_init_pwm + (-wiggle_middle)*HLB_direction)
158.         goal_dict['HLM'] = int(HLM_init_pwm + (wiggle_v - HL_height)*HLM_direction)

159.         goal_dict['HLE'] = int(HLE_init_pwm + (wiggle_v + 0)*HLE_direction)
160.     elif pos == 2:
161.         goal_dict['HLB'] = int(HLB_init_pwm + (-wiggle_middle + wiggle_h*direction_in
        put)*HLB_direction)
162.         goal_dict['HLM'] = int(HLM_init_pwm - HL_height*HLM_direction)
163.         goal_dict['HLE'] = int(HLE_init_pwm)
164.     else:
165.         goal_dict['HLB'] = int(HLB_init_pwm + (-wiggle_middle + (wiggle_h*(6-(pos-2))
        /3 -

```

```

166.                                     wiggle_h)*direction_
    input)*HLB_direction)
167.         goal_dict['HLM'] = int(HLM_init_pwm - HL_height*HLM_direction)
168.         goal_dict['HLE'] = int(HLE_init_pwm)
169.         #print('HL: %d'%pos)
170.
171. def leg_HR(pos, direction_input):
172.     ....
173.         Entering the pos parameter to generate the PWM value of the leg at that position,
            it is the gait generation method of the left hind leg.
174.         ...
175.         if pos == 1:
176.             goal_dict['HRB'] = int(HRB_init_pwm + (-wiggle_middle)*HRB_direction)
177.             goal_dict['HRM'] = int(HRM_init_pwm + (wiggle_v - HR_height)*HRM_direction)
178.             goal_dict['HRE'] = int(HRE_init_pwm + (wiggle_v + 0)*HRE_direction)
179.         elif pos == 2:
180.             goal_dict['HRB'] = int(HRB_init_pwm + (-wiggle_middle + wiggle_h*direction_in
                put)*HRB_direction)
181.             goal_dict['HRM'] = int(HRM_init_pwm - HR_height*HRM_direction)
182.             goal_dict['HRE'] = int(HRE_init_pwm)
183.         else:
184.             goal_dict['HRB'] = int(HRB_init_pwm + (-wiggle_middle + (wiggle_h*(6-(pos-2))
                /3 -
185.                                     wiggle_h)*direction_
                input)*HRB_direction)
186.             goal_dict['HRM'] = int(HRM_init_pwm - HR_height*HRM_direction)
187.             goal_dict['HRE'] = int(HRE_init_pwm)
188.             #print('HR: %d'%pos)
189.             #print(position_input)
190.             if gait_set == 0 or now_command == 'turnleft' or now_command == 'turnright':
191.                 ....
192.                 Diagonal gait is generated here. You can refer to the diagonal gait table shown e
                    arlier in this chapter for the generation method.
193.                 ...
194.                 if position_input == 1:
195.                     leg_FL(1, left_direction)
196.                     leg_FR(5, right_direction)
197.
198.                     leg_HL(5, left_direction)
199.                     leg_HR(1, right_direction)
200.                 pass

```



```

201.         elif position_input == 2:
202.             leg_FL(2, left_direction)
203.             leg_FR(8, right_direction)
204.
205.             leg_HL(8, left_direction)
206.             leg_HR(2, right_direction)
207.             pass
208.         elif position_input == 5:
209.             leg_FL(5, left_direction)
210.             leg_FR(1, right_direction)
211.
212.             leg_HL(1, left_direction)
213.             leg_HR(5, right_direction)
214.             pass
215.         elif position_input == 8:
216.             leg_FL(8, left_direction)
217.             leg_FR(2, right_direction)
218.
219.             leg_HL(2, left_direction)
220.             leg_HR(8, right_direction)
221.             pass
222.     elif gait_set == 1:
223.         ....
224.         The triangle gait is generated here. You can refer to the triangle gait table shown
           earlier in this chapter for the generation method.
225.         ...
226.         if position_input == 1:
227.             leg_FL(1, left_direction)
228.             leg_FR(5, right_direction)
229.
230.             leg_HL(3, left_direction)
231.             leg_HR(7, right_direction)
232.             pass
233.         elif position_input == 2:
234.             leg_FL(2, left_direction)
235.             leg_FR(6, right_direction)
236.
237.             leg_HL(4, left_direction)
238.             leg_HR(8, right_direction)
239.             pass
240.         elif position_input == 3:
241.             leg_FL(3, left_direction)

```

```
242.         leg_FR(7, right_direction)
243.
244.         leg_HL(5, left_direction)
245.         leg_HR(1, right_direction)
246.         pass
247.     elif position_input == 4:
248.         leg_FL(4, left_direction)
249.         leg_FR(8, right_direction)
250.
251.         leg_HL(6, left_direction)
252.         leg_HR(2, right_direction)
253.         pass
254.     elif position_input == 5:
255.         leg_FL(5, left_direction)
256.         leg_FR(1, right_direction)
257.
258.         leg_HL(7, left_direction)
259.         leg_HR(3, right_direction)
260.         pass
261.     elif position_input == 6:
262.         leg_FL(6, left_direction)
263.         leg_FR(2, right_direction)
264.
265.         leg_HL(8, left_direction)
266.         leg_HR(4, right_direction)
267.         pass
268.     elif position_input == 7:
269.         leg_FL(7, left_direction)
270.         leg_FR(3, right_direction)
271.
272.         leg_HL(1, left_direction)
273.         leg_HR(5, right_direction)
274.         pass
275.     elif position_input == 8:
276.         leg_FL(8, left_direction)
277.         leg_FR(4, right_direction)
278.
279.         leg_HL(2, left_direction)
280.         leg_HR(6, right_direction)
281.         pass
```