Sheepdog Project Report

Dingming Lu

ME 375

04/28/2023

I. Introduction

With the knowledge we have learned in control class, we used PID controllers and close loop feedback to make it possible. In this project, we will let the robot follow the line and run for one lap, then it will switch to herding mode automatically and keep the target at a distance of 9 inches. Once we let it go, it's all on its own. In this report, I will discuss how we design the controllers, how the robot performed, the code make it run, and some problems we have encountered.

II. Controller Design

A. Speed Control Design

In previous labs, we first measured the system's time constant and static gain by inputting a square wave signal. It ran forward for four seconds and backward for four seconds. Based on the reading from the encoder, we can roughly measure and calculate the time constant and the static gain (see table 2.1.1). Deadband voltages are also measured.

Table 2.1.1 Time constant, the static gain, and the deadband voltage of the robot

	Time constant(sec)	Static Gain	Deadband (V)
Left wheel	0.23	3.617	3.3
Right wheel	0.2	4.118	4.18

Based on the value from table 2.1.1 and direct pole-placement method with 0.1 second of 2% settling time, Kp and Ki are calculated for each wheel (see table 2.1.2)

Table 2.1.2 Kp and Ki values for both wheels.

	Left wheel	Right Wheel
Kp	4.21	3.58
Ki	256.96	218.64

Now the values of the PI controller are determined. We put it into the block diagram and the steady state error is eliminated as expected.

With the values in table 2.1.2, the robot has high-frequency oscillation when it's running, so we decreased Kp and Ki to 1.5 and 40 and there was no shaking anymore.

However, we we were trying to make it run in the following line task, we have to change Kp to 0.7 and Ki to 5 so the robot didn't run off the course and ran smoothly.

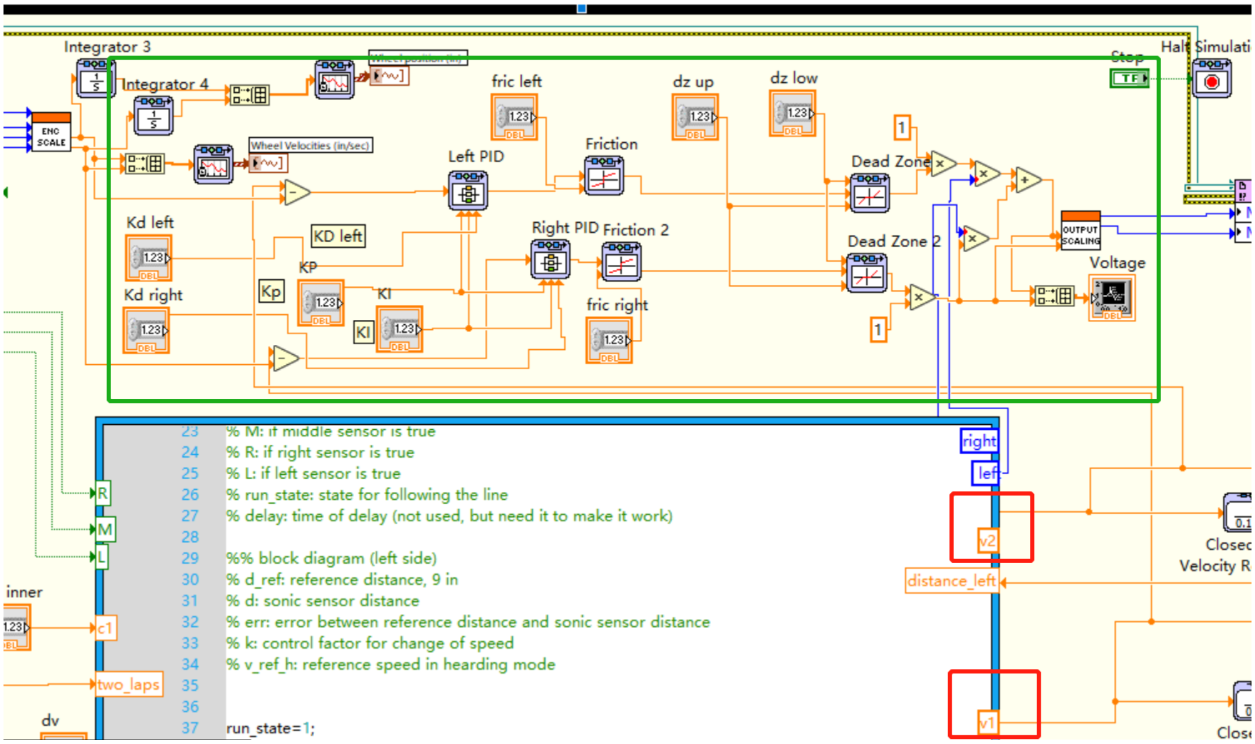

Figure 2.1.1 Speed control block diagram. Making numeric control to the front panel can k the tunning easier and faster. The red boxes are the input of the controller (from the output of the code), and the green box is the controller.

B. Follow the Line Controller Design

When the detector detects the black line, it returns a value of “true”. When the right sensor is true, it should turn right, vice versa. Based on this logic, we separate the control into five parts (see table 2.2.1)

Table 2.2.1 True False Table of the Controller.

Left	Middle	Right	Control Path
0	0	0	Reverse the previous input
0	0	1	Hard right
0	1	0	Straight
0	1	1	Right
1	0	0	Hard left
1	0	1	Straight
1	1	0	Left
1	1	1	Straight

When it’s turning, we make the inner wheel spin slower, and the outer wheel spin faster. We call the change of speed as “dv”. When it’s in hard turn, we give dv a factor “c1” and “c2” for the inner wheel and the outer wheel so it changes even more from the reference speed (see table 2.2.2). For the left wheel, the change of speed is dv_l, for the right wheel, change of speed is dv_r. The change of speed for the inner wheel is -dv, outer wheel is +dv.

Table 2.2.2 Change of speed is based on states.

Control Path	Left Wheel Speed (v2)	Right Wheel Speed (v1)
Straight	v_ref	v_ref
Left	v_ref+dv_l, dv_l=-dv	v_ref+dv_r, dv_r=dv
Hard Left	v_ref+c1*dv_l, dv_l=-dv	v_ref+c2*dv_r, dv_r=dv
Right	v_ref+dv_l, dv_l=dv	v_ref+dv_r, dv_r=-dv
Hard Right	v_ref+c2*dv_l, dv_l=dv	v_ref+c1*dv_r, dv_r=-dv

Then we put this control statements into the code as “if” statements. Based on the true-false table and the distance it traveled, it will run a different command.

```
% forward
elseif ((M==1) & (R==0) & (L==0)) | ((M==1) & (R==1) & (L==1)) | ((M==0) & (R==1) & (L==1)) & (distance_l
v1 = v_ref;
v2 = v_ref;
right=0;
left=1;
i=1;
next_state=run_state;
```

```
% right
elseif (M==1) & (R==1) & (L==0) &(distance_left<two_laps)
 dv_l=dv;
 dv_r=-dv;
 v1 = v_ref+dv_r;
 v2 = v_ref+dv_l;
right=0;
left=1;
i=2;
next_state=run_state;
```

```
%hard right
elseif (M==0) & (R==1) & (L==0) &(distance_left<two_laps)
dv_l=dv;
dv_r=-dv;
v1 = v_ref+c1*dv_r;
v2 = v_ref+c2*dv_l;
right=0;
left=1;
i=3;
next_state=run_state;
```

```
% Left
elseif (M==1) & (R==0) & (L==1) &(distance_left<two_laps)
dv_l=-dv;
dv_r=dv;
v1 = v_ref+dv_r;
v2 = v_ref+dv_l;
right=0;
left=1;
i=4;
next_state=run_state;

% Hard Left
elseif (M==0) & (R==0) & (L==1) & (distance_left<two_laps)
 dv_l=-dv;
 dv_r=dv;
 v1 = v_ref+c2*dv_r;
 v2 = v_ref+c1*dv_l;
right=0;
left=1;
i=5;
next_state=run_state;

end
```

Figure 2.2.1 Code of following line controller

C. Herding Controller Design

In this task, we will let the robot target 9 inches away from an object. The feedback if from the sonic sensor. When it's too close, the robot should go reverse; when the target is too far away, it will go forward. We multiply the error by a constant and set them to the reference speed. The error is calculated by the actual distance from the sonic sensor minus the reference distance of 9 inches. When it's too close, the error is negative so it will go reverse. To limit the output so it doesn't go crazy, we set the upper limit and the lower limit to 6. This can prevent unstable control when the terrible sonic sensor accidentally measured crazy high outlines.

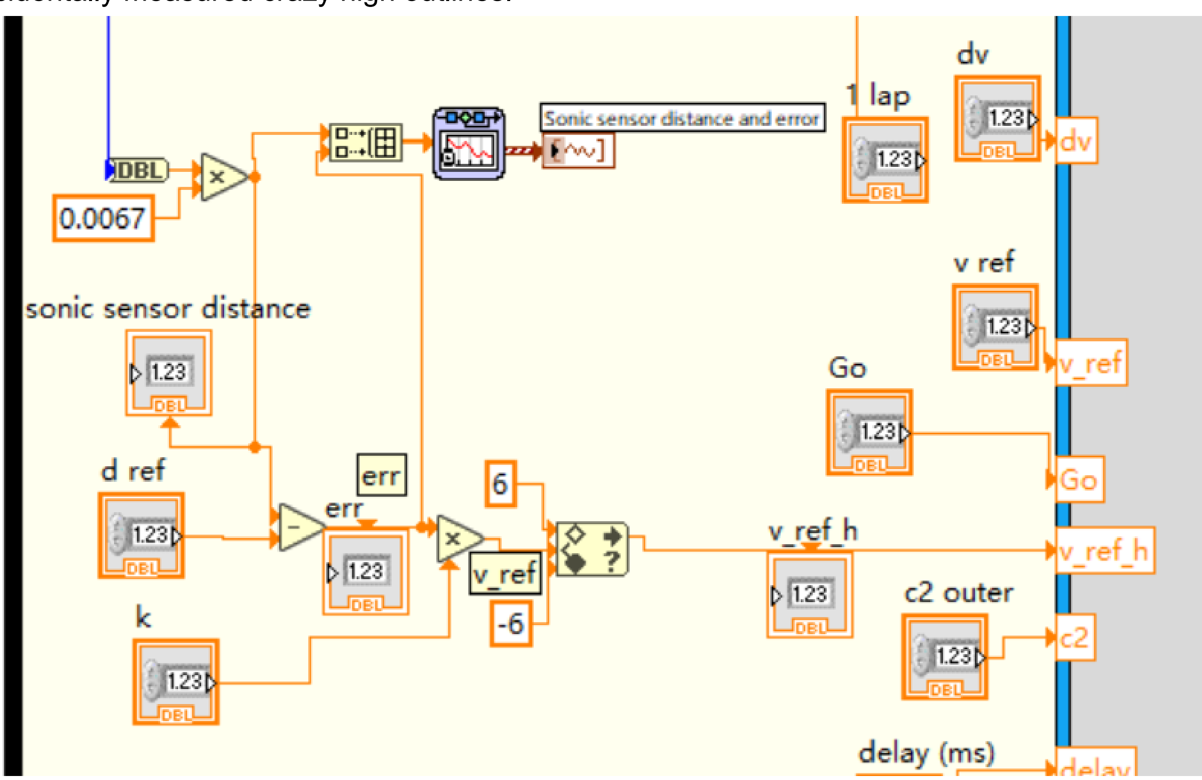

Figure 2.3.1: Controller or the herding.

We need some addition code to switch the state from following the line mode to the herding mode. We have to set the speed to zero before entering the herding mode, and force the output of the PID speed controller of the right wheel directly into the left wheel. This will be discussed in later section.

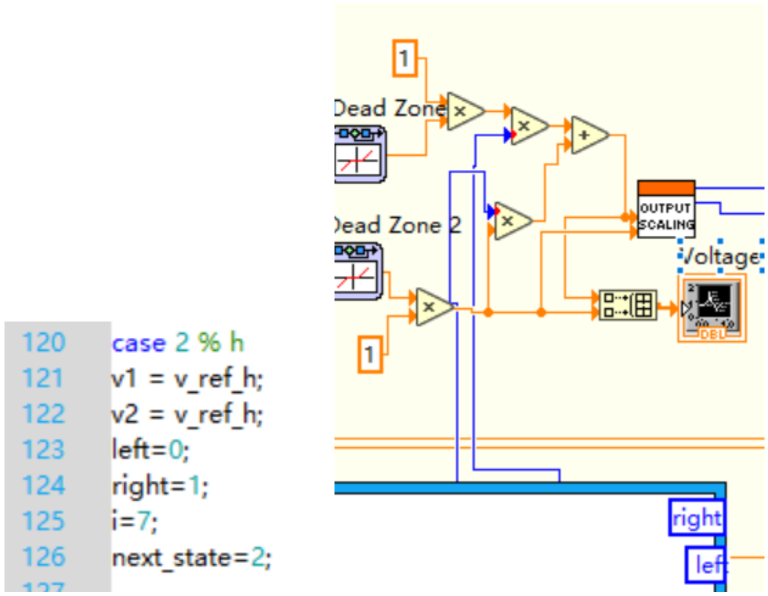

Figure 2.3.2: Code and the right wheel signal to the left wheel. The building process of this section will be discussed later on.

III. Performance

A. Starting State

When the “Go” becomes true, it will change it to the running state.

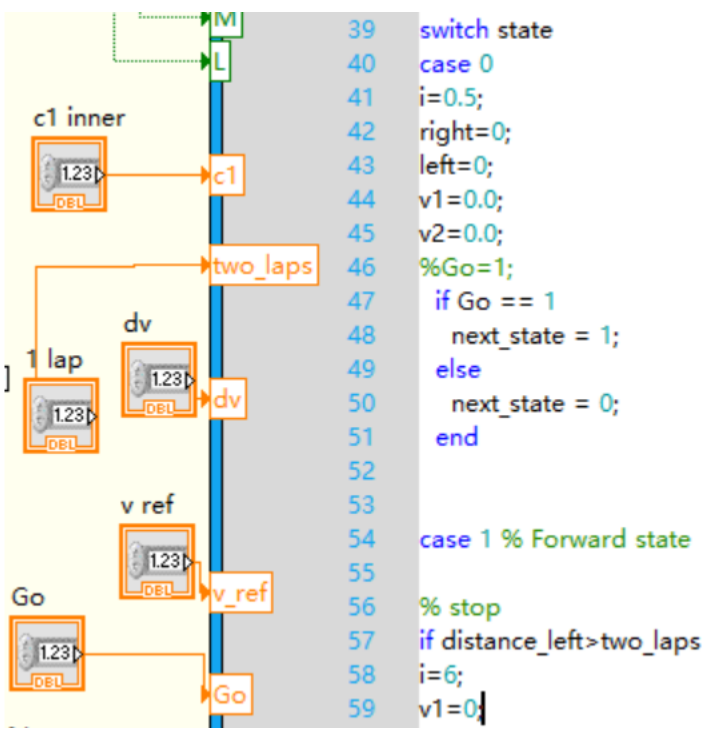

Figure 3.1.1: The code for change from starting state to forward state. The reason why “Go” is not a green true-false button will be discussed later on.

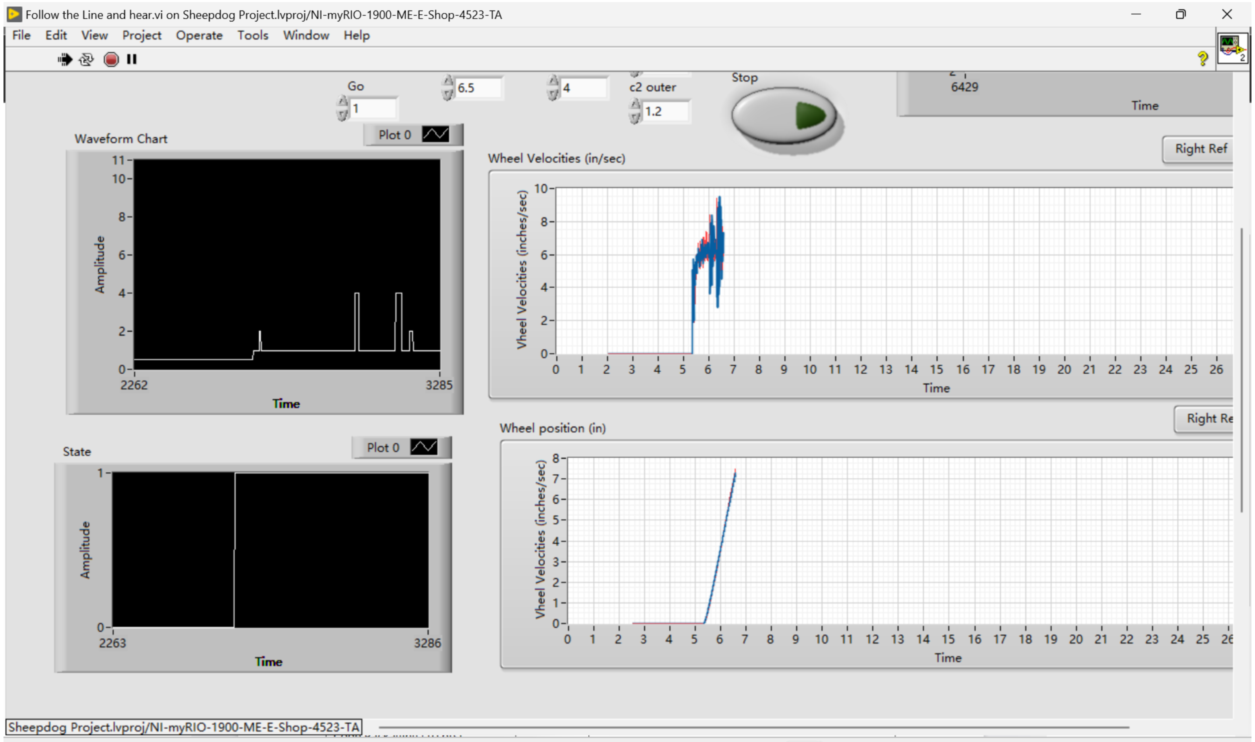

Figure 3.1.2: Once “Go” becomes true (or 1), it will change the state from zero to 1 (lower left corner is the state indicator), the wheels started spinning (upper right graph), and the control path of following lines started working (different number indicates different control path).

B. Following The Line

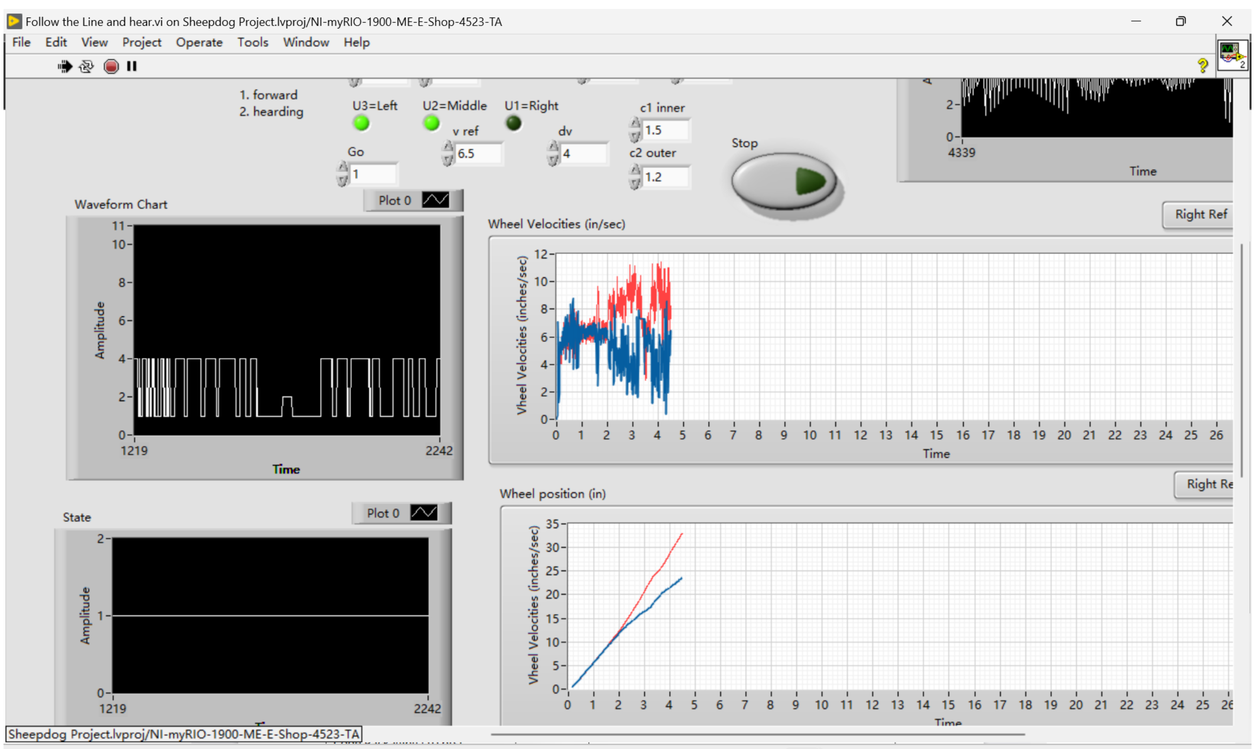

Figure 3.2.1: In the upper left graph, when it’s turing left, the control path is switching between 1 (straight) and 4 (left). In the upper right graph, the right wheel spins faster. In the lower left graph, it’s always in the following line state. In the lower right graph, the right wheel travels longer.

C. Herding

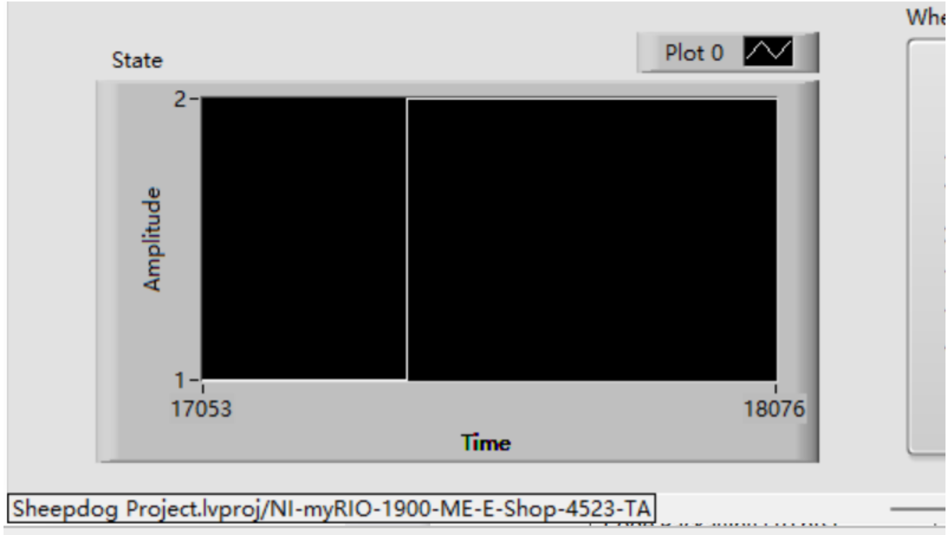

Figure 3.3.1: Changing following line mode (state 1) to herding mode (state 2).

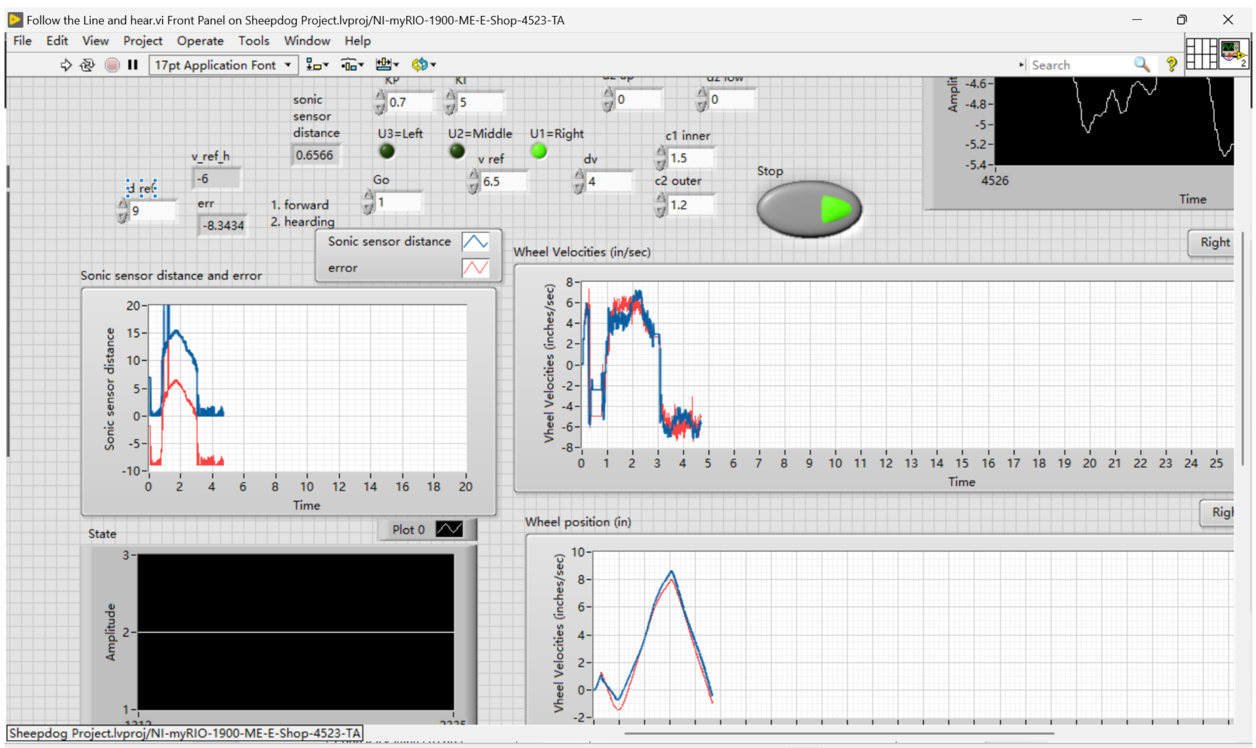

Figure 3.3.2: In the upper left graph, the blue line is the actual distance measured by the sonic sensor, and the red line is the error. In the upper right graph, when the error is positive from 1 second to 3 seconds, it means the target is too far, the speed of the wheel is positive, and the displacement (lower right graph) is increasing. When the error is negative from 3 seconds to 5 seconds, the speed of the wheel is negative, and the displacement (lower right graph) is decreasing.

IV. Code Description

A. Overview

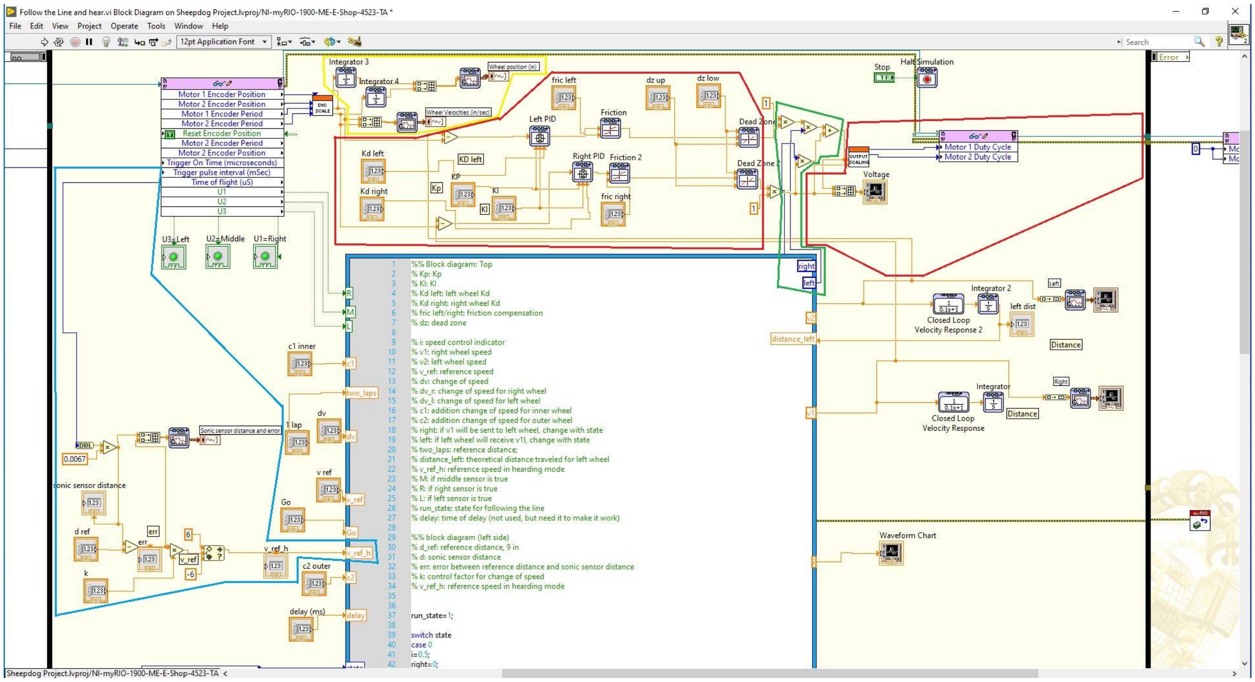

Figure 4.1.1: Overview of the block diagram.
Yellow box on the top is the speed and position indicator.
Red boxes are the speed control.
Green box is the special switching control for herding
Blue box is the reference speed control for the herding mode.

B. Speed Control

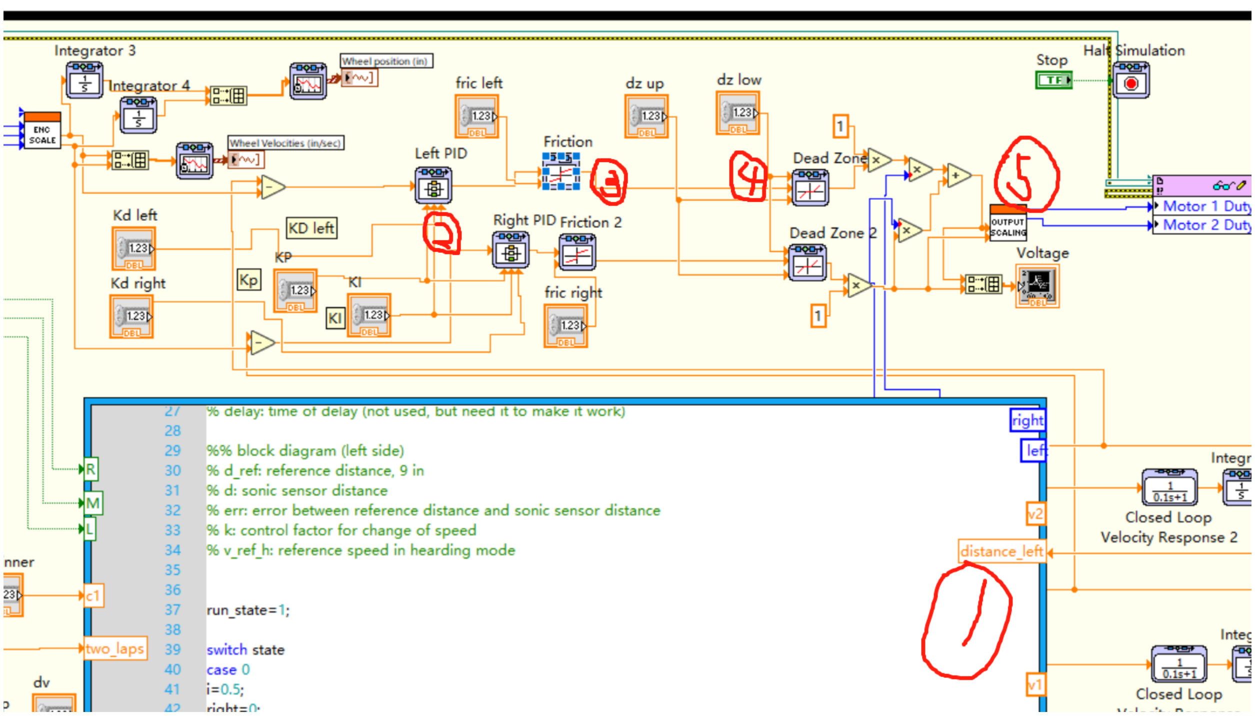

Figure 4.2.1: Speed control. v1 and v2 will be compared with the actual wheel speed (1), and put the difference into the PID controller (2), then add friction compensation (3) and deadzone (4) to eliminate high frequency small amplitude shaking, finally output the value to the motor drive (5). The values for the friction compensator and the dead zone limits are zero in this situation.

C. Following The Line

All the codeshare in the blue box at the center of the VI. The language used is MathScript.

Line 1 to 34 describes what each variable means.

```
1  %% Block diagram: Top
2  % Kp: Kp
3  % KI: KI
4  % Kd left: left wheel Kd
5  % Kd right: right wheel Kd
6  % fric left/right: friction compensation
7  % dz: dead zone
8
9  % i: speed control indicator
10 % v1: right wheel speed
11 % v2: left wheel speed
12 % v_ref: reference speed
13 % dv: change of speed
14 % dv_r: change of speed for right wheel
15 % dv_l: change of speed for left wheel
16 % c1: addition change of speed for inner wheel
17 % c2: addition change of speed for outer wheel
18 % right: if v1 will be sent to left wheel, change with state
19 % left: if left wheel will receive v1l, change with state
20 % two_laps: reference distance;
21 % distance_left: theoretical distance traveled for left wheel
22 % v_ref_h: reference speed in hearding mode
23 % M: if middle sensor is true
24 % R: if right sensor is true
25 % L: if left sensor is true
26 % run_state: state for following the line
27 % delay: time of delay (not used, but need it to make it work)
28
29 %% block diagram (left side)
30 % d_ref: reference distance, 9 in
31 % d: sonic sensor distance
32 % err: error between reference distance and sonic sensor distance
33 % k: control factor for change of speed
34 % v_ref_h: reference speed in hearding mode
```

Line 37: assign the value for the running state

Line 40 to 51: starting state

```
35
36
37 run_state=1;
38
39 switch state
40 case 0
41 i=0.5;
42 right=0;
43 left=0;
44 v1=0.0;
45 v2=0.0;
46 %Go=1;
47   if Go == 1
48     next_state = 1;
49   else
50     next_state = 0;
51   end
52
53
```

Line 54: following the line state
Line 56 to 63: if the robot finish one lap distance, it will get into the herding mode.
Line 65 to 72: go forward, if it didn’t finish one lap yet, and the condition for going forward is met (refer to table 2.2.1 and table 2.2.2)

```
54 case 1 % Forward state
55
56 % stop
57 if distance_left>two_laps
58 i=6;
59 v1=0;
60 v2=0;
61 right=0;
62 left=1;
63 next_state=2;
64
65 % forward
66 elseif ((M==1) & (R==0) & (L==0)) | ((M==1) & (R==1) & (L==1)) | ((M==0) & (R==1) & (L==1)) & (distance_l
67 v1 = v_ref;
68 v2 = v_ref;
69 right=0;
70 left=1;
71 i=1;
72 next_state=run_state;
73
```

Line 74 to 83: turn right (refer to table 2.2.1 and table 2.2.2)
Line 85 to 94: turn hard right (refer to table 2.2.1 and table 2.2.2)
Line 96 to 105: turn left (refer to table 2.2.1 and table 2.2.2)
Line 107 to 116: turn hard left (refer to table 2.2.1 and table 2.2.2)

```
74 % right
75 elseif (M==1) & (R==1) & (L==0) &(distance_left<two_laps)
76  dv_l=dv;
77  dv_r=-dv;
78  v1 = v_ref+dv_r;
79  v2 = v_ref+dv_l;
80 right=0;
81 left=1;
82 i=2;
83 next_state=run_state;
84
85 %hard right
86 elseif (M==0) & (R==1) & (L==0) &(distance_left<two_laps)
87 dv_l=dv;
88 dv_r=-dv;
89 v1 = v_ref+c1*dv_r;
90 v2 = v_ref+c2*dv_l;
91 right=0;
92 left=1;
93 i=3;
94 next_state=run_state;
95
```

```
% Left
elseif (M==1) & (R==0) & (L==1) &(distance_left<two_laps)
dv_l=-dv;
dv_r=dv;
v1 = v_ref+dv_r;
v2 = v_ref+dv_l;
right=0;
left=1;
i=4;
next_state=run_state;

% Hard Left
elseif (M==0) & (R==0) & (L==1) & (distance_left<two_laps)
 dv_l=-dv;
 dv_r=dv;
 v1 = v_ref+c2*dv_r;
 v2 = v_ref+c1*dv_l;
right=0;
left=1;
i=5;
next_state=run_state;

end
```

D. Forcing Control Signal into Left Wheel

We did this because the encoder in the left wheel has some problem. We can use the linear combination to connect the output of the right wheel PID control into the left wheel: left_wheel_speed=left*v2+right*v1, where v2 is the left wheel speed, and v1 is the right wheel speed. When left is 1, right is 0, the output is v2. When left is 0, right is 1, the output is v1.

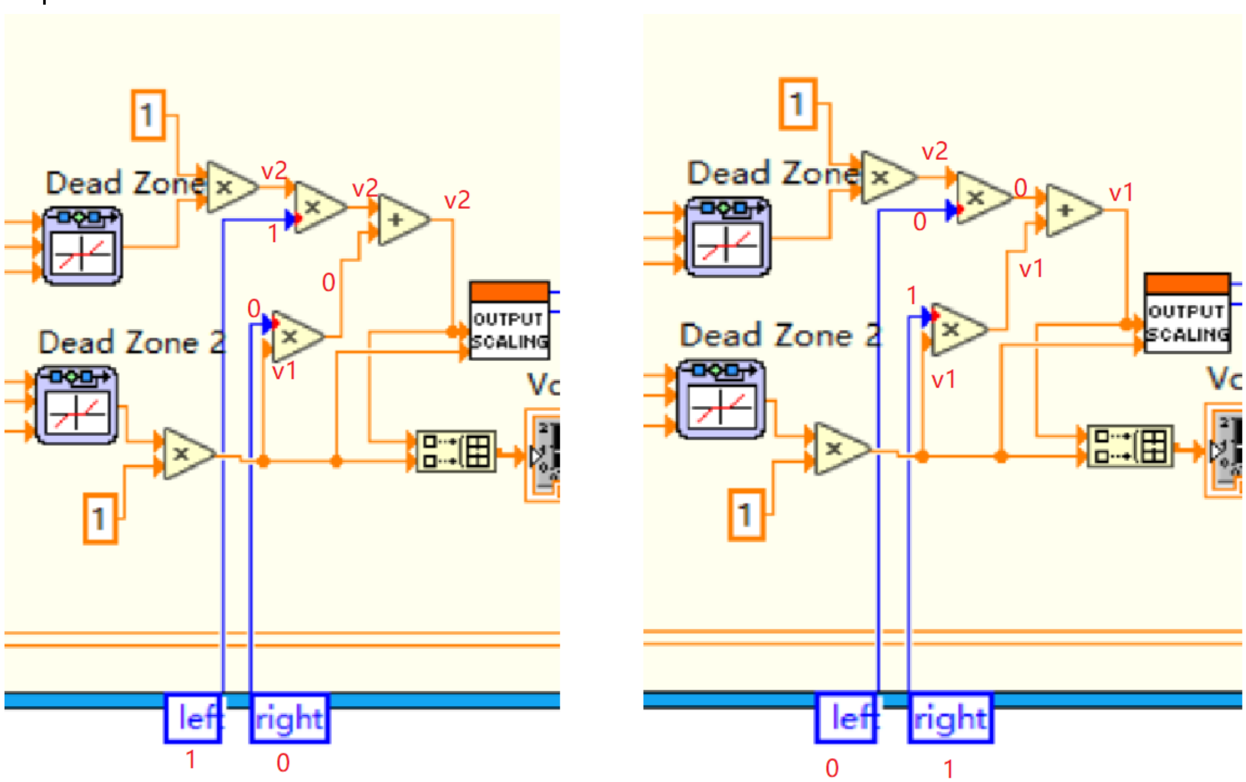

Figure 4.4.1: Left: In the following line mode, set left to 1 and right to 0, output is left wheel speed into left wheel. Right: In the herding mode, set left to 0 and right to 1, output is right wheel speed into left wheel.

E. Herding

We used the block diagram majority because the code sometimes works and is not stable. Many bugs were fixed by using the block diagram. It will take the measurement of flight time from the sonic sensor and multiply by a constant to conver it into distance (1). Then it compares the reference distance to find the error (2). Multiply the error (3) by a factor and limit the output (4), the value becomes the reference speed (5). The reference speed can be positive or negative depends on the error.

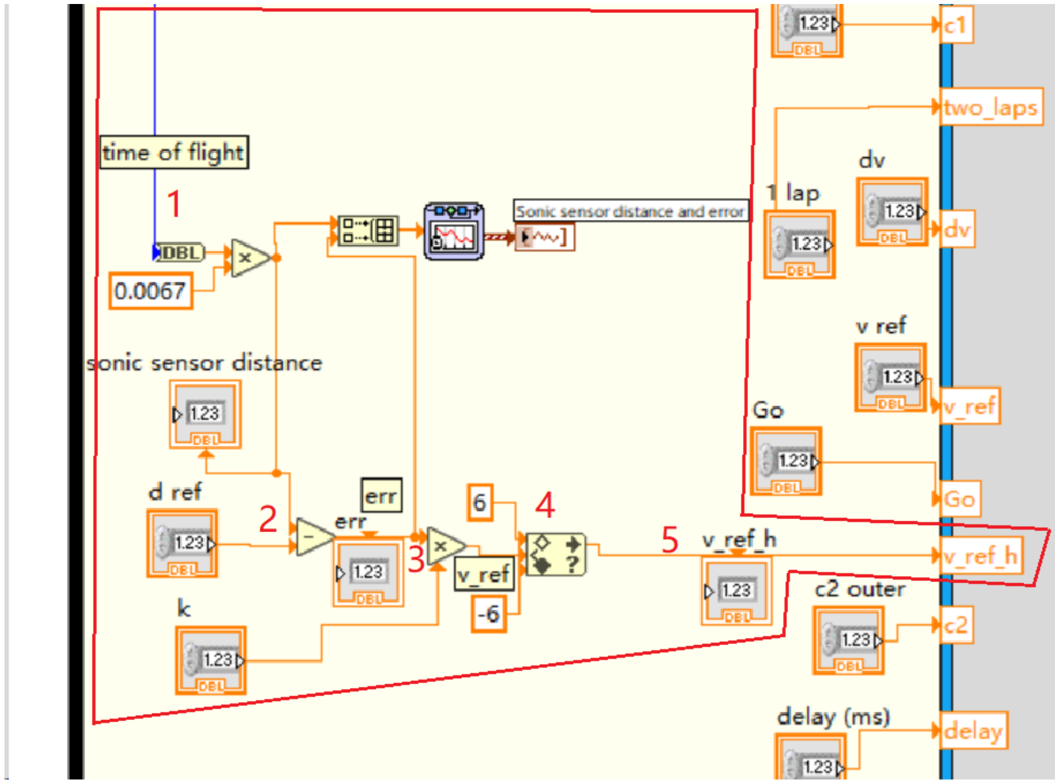

Figure 4.5.1: Block diagram for herding.

Then it will assign the speed for the left wheel and right wheel. Although v2 is not really useful according to figure 4.4.1, we still need to keep it in the code, otherwise, the code won't run.

```
case 2 % h
v1 = v_ref_h;
v2 = v_ref_h;
left=0;
right=1;
i=7;
next_state=2;
```

Figure 4.5.2: the code for herding

V. Discussion

A. Problem-Solving Part 1

a. High-frequency oscillation during controller design
During the controller design, the robot had sudden stops. It's not going smoothly. We fixed this problem by decreasing Kp and Ki, meanwhile keeping Kp and Ki as high as possible.

b. High-frequency oscillation during the Following Line testing
Even though the problem was solved in previous labs, the wheel started to shake again. This can be caused by difference reference speed. In the previous section,

the Kp and Ki works fine with the reference speed of 10. But in this taks, the reference speed is around 6, so I have to change Kp and Ki accordingly because the actual robot is not linear and Kp and Ki could be changed with the speed according to Dr. Lillian. I decreased Kp to 0.7 and Ki to 5, and finally it can run smoothly.

B. Problem-Solving Part 2

The purpose of this section is to collect all the “everything looks right but nobody even GTA and Dr. Lillian don’t know how to fix it” problems.

a. **Problem**: the code is not running the command when the condition is true. When the distance traveled is greater than one lap, it should change the state, but it’s not.
Solution: we have to put it into the first “if” statement (line 57). It just won’t work in “elseif”.
b. **Problem**: after assigning new variables “right” and “left” into the running state (line 61 and 62, etc), the entire code is not running.
Solution: add the new variables into every states, even though they are not necessary (line 42 to 43).
c. **Problem**: even “Go” equals to 1, it doesn’t want to change state (line 47 to 48).
Solution: Unknown. Somehow it works. To see if “Go: truly becomes 1, we change the button into a numeric control and add the indicator.
d. **Problem**: in herding mode, the left wheel is not behaving correctly. It has high-frequency oscillation. Except for the reference speed, everything else in the speed control is the same. Even more strangely, the right wheel behaves correctly.
Solution: Use the same signal input as the right wheel. I don’t think the encoder is broken because it was totally fine in the following line mode.
e. **Problem**: The Sonic sensor is measuring 0.01 inch or more than 600 inches, causing the odd behavior even though everything is correct, such as when the indicator is showing 0.03 inches, the error is negative, the robot was still trying to go forward.
Solution: change to a new sonic sensor, or move the target slowly.
f. **Problem**: The robot ran off the course at that specific track and that specific corner.
Solution: use another track, that one has some lighting issues.
g. **Problem**: after adding the reverse control when the robot is off the course (first condition in table 2.1.1), the code behaves like part c. Nothing in the blue script box is running.
Solution: get rid of that control. The previous value is not passing into the next loop because the entire code won’t run for some reason.

C. Robot Performance

a. Following the line
The robot ran smoothly on the straight line. When it’s on the curve, it had some sudden turns. This is caused by not having enough line sensor elements, and the deflection is very large when they detect something.
b. Herding
There’s a fifty-fifty chance that the sonic sensor won’t behave correctly. From the indicator, the distance it measured can have large outlines. This causes wrong error calculation and wrong speed input.
When the sonic sensor was working, it behaves correctly (refer to figure 3.3.2)

VI. Appendix

Full code

```
%% Block diagram: Top
% Kp: Kp
% KI: KI
% Kd left: left wheel Kd
% Kd right: right wheel Kd
% fric left/right: friction compensation
% dz: dead zone

% i: speed control indicator
% v1: right wheel speed
% v2: left wheel speed
% v_ref: reference speed
% dv: change of speed
% dv_r: change of speed for right wheel
% dv_l: change of speed for left wheel
% c1: addition change of speed for inner wheel
% c2: addition change of speed for outer wheel
% right: if v1 will be sent to left wheel, change with state
% left: if left wheel will receive v1l, change with state
% two_laps: reference distance;
% distance_left: theoretical distance traveled for left wheel
% v_ref_h: reference speed in hearding mode
% M: if middle sensor is true
% R: if right sensor is true
% L: if left sensor is true
% run_state: state for following the line
% delay: time of delay (not used, but need it to make it work)

%% block diagram (left side)
% d_ref: reference distance, 9 in
% d: sonic sensor distance
% err: error between reference distance and sonic sensor distance
% k: control factor for change of speed
% v_ref_h: reference speed in hearding mode


run_state=1;

```

```
switch state
case 0
i=0.5;
right=0;
left=0;
v1=0.0;
v2=0.0;
%Go=1;
  if Go == 1
    next_state = 1;
  else
    next_state = 0;
  end


case 1 % Forward state % For the logic, refer to table 2.1.1

% stop
if distance_left>two_laps
i=6;
v1=0;
v2=0;
right=0;
left=1;
next_state=2;

% forward
elseif ((M==1) & (R==0) & (L==0)) | ((M==1) & (R==1) & (L==1)) | ((M==0) & (R==1) & (L==1)) & (distance_left<two_laps)
v1 = v_ref;
v2 = v_ref;
right=0;
left=1;
i=1;
next_state=run_state;

```

```
% right
elseif (M==1) & (R==1) & (L==0) &(distance_left<two_laps)
 dv_l=dv; % addition speed (+) for left wheel
 dv_r=-dv; % addition speed (-) for right wheel
 v1 = v_ref+dv_r; % addition speed (-) for right wheel
 v2 = v_ref+dv_l; % addition speed (+) for left wheel
right=0;
left=1;
i=2;
next_state=run_state;

%hard right
elseif (M==0) & (R==1) & (L==0) &(distance_left<two_laps)
dv_l=dv;
dv_r=-dv;
v1 = v_ref+c1*dv_r; % more addition speed (-) for right wheel
v2 = v_ref+c2*dv_l; % more addition speed (+) for left wheel
right=0;
left=1;
i=3;
next_state=run_state;

% Left
elseif (M==1) & (R==0) & (L==1) &(distance_left<two_laps)
dv_l=-dv;
dv_r=dv;
v1 = v_ref+dv_r;
v2 = v_ref+dv_l;
right=0;
left=1;
i=4;
next_state=run_state;

% Hard Left
elseif (M==0) & (R==0) & (L==1) & (distance_left<two_laps)
 dv_l=-dv;
 dv_r=dv;
 v1 = v_ref+c2*dv_r;
 v2 = v_ref+c1*dv_l;
right=0;
left=1;
i=5;
next_state=run_state;

end

case 2 % herding mode
v1 = v_ref_h;
v2 = v_ref_h;
left=0;
right=1;
i=7;
next_state=2;

end
```