山东大学

崇新学堂

2025－2026 学年第一学期

实 验 报 告

课程名称：电子信息工程导论

实验名称：Controlling Robots

专 业 班 级　崇新学堂

学 生 姓 名　高子轩，钱竹玉，吕思洁，徐亚骐

实 验 时 间　2025 年 9 月 24 日

Step1： Sonars Test：

We uncommented the line print inp.sonars[3], and set the robot to a stationary state, only focusing on whether the sonar can work normally.

The specific code is as follows:

```
1. class MySMClass(sm.SM):
2.     def getNextValues(self, state, inp):
3.         return (state, io.Action(fvel=0, rvel=0))
```

```
1. def setup():
2.     robot.gfx = gfx.RobotGraphics(drawSlimeTrail=True, # slime trails
3.                                   sonarMonitor=True) # sonar monitor widget
```

```
1. def step():
2.     inp = io.SensorInput()
3.     print inp.sonars[3]
4.     robot.behavior.step(inp).execute()
5.     io.done(robot.behavior.isDone())
```

Checkoff 1.Explain to a staff member the results of your experiments with the sonars. Demonstrate that you know your partner's name and email address.

Both the rotational speed and the forward speed are set to 0, so the robot does not perform any actions including rotation and forward movement. We set the "sonar Monitor" to be "True" , printing the sonar sensor value through a for loop. The robot cyclically detects the input signal of the sonar in each step and

outputs it through the print function.

The key code is as follows:

```
1. robot.gfx = gfx.RobotGraphics(drawSlimeTrail=True,sonarMonitor=True)

2. print("--- Sonar Readings ---")

3. for i in range(len(inp.sonars)):

4.     print("Sonar {0}: {1:.2f} meters".format(i, inp.sonars[i]))

5. print("---------------------\n")
```

The output image is as follows:

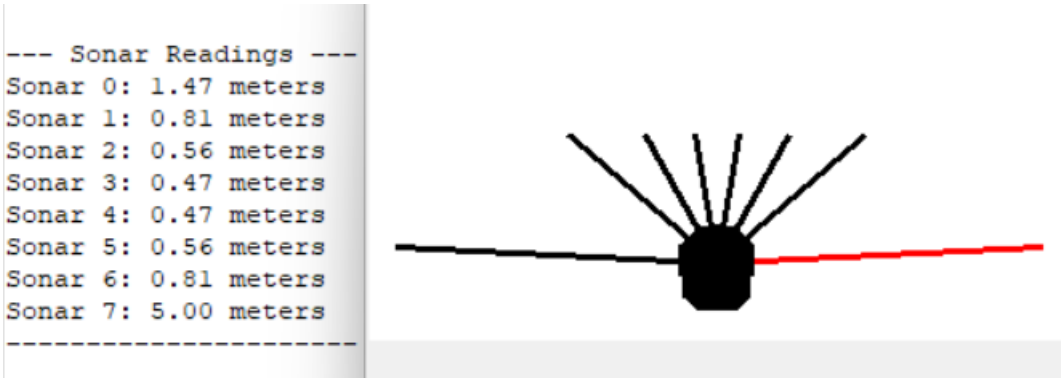

Figure 1 Visualization of sonar sensor data.

By doing this, we confirmed：

1. The vehicle's entire sonar system is functioning properly.

2. The maximum effective range of the sonar is 1.5 meters.

3. When the sonar is too close to an obstacle, the readings become highly unstable.

Checkoff 2. Wk.2.2.2: Demonstrate your distance-keeping brain on a real robot to a staff member.

Code logic We set the target_distance to 0.5,and get the front_distance through sonars[3].We set error to target_distance - front_distance.When the error is relatively huge,the speed should be faster.In conclusion,speed and error exhibit a linear relationship, and the degree of this correlation can be adjusted by modifying the parameters.

The specific code is as follows:

```
class MySMClass(sm.SM):
    def getNextValues(self, state, inp):
        target_distance = 0.5
        front_distance = inp.sonars[3]
        error = front_distance - target_distance
        # Compute forward speed using a proportional controller (scaled by 0.1)
        fvel = error * 0.1
        # Limit the forward speed from -0.3 to 0.3 to prevent excessive movement
        if fvel > 0.3:
            fvel = 0.3
        elif fvel < -0.3:
            fvel = -0.3
        rvel = 0.0
        # Return the current state and the movement action
        return (state, io.Action(fvel=fvel, rvel=rvel))
```

Simulation results:

The robot successfully moved to a place approximately 0.5 meters away

from the obstacle in front of it and still maintained stability when the environment was changed

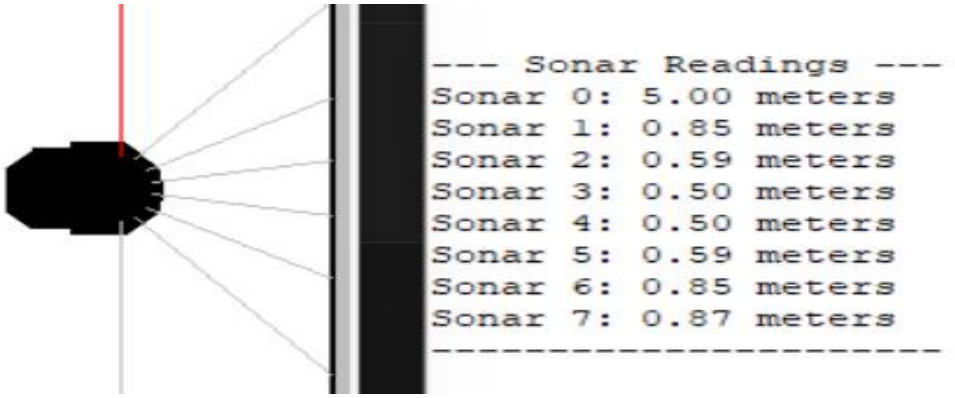

Figure 2 Final distance-keeping result.

Wk.2.2.3: Show your state-transition diagram to a staff member. Make clear what the conditions on state transitions are, and what actions are as sociated with each state.

State transition diagram:

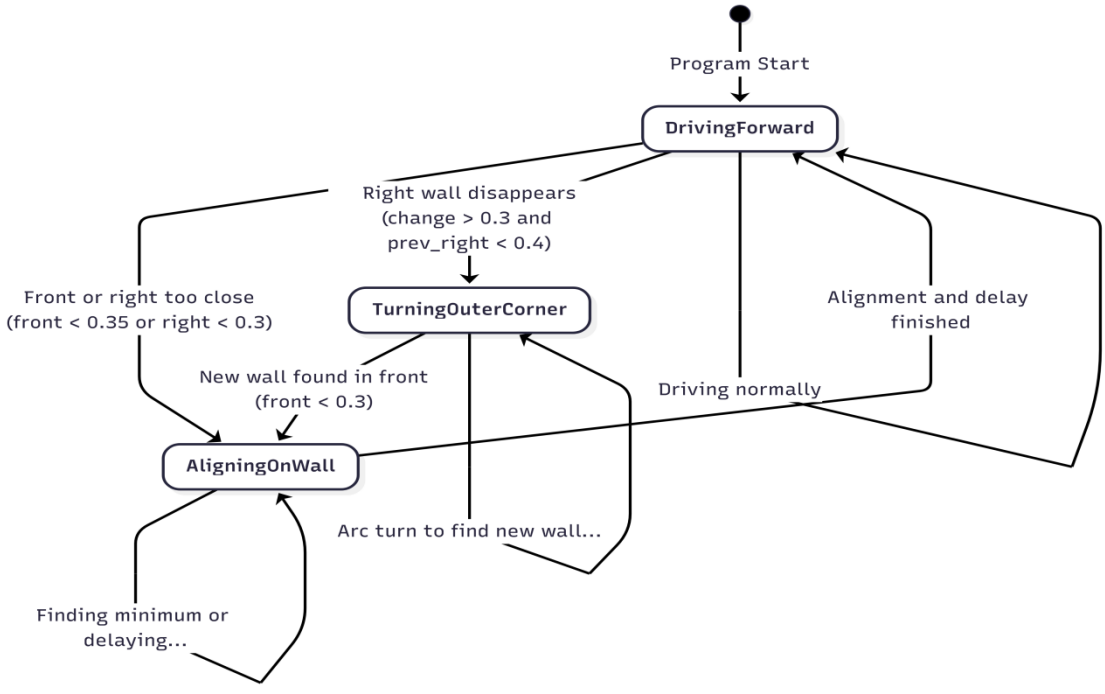

Figure 3 Final state transition diagram

Here is the general working logic:

• Initially, it is in a straight state. Set the smaller value of the readings of Sonar[3] and Sonar[4] to the distance to the front. For the program that walks along the wall, when this distance is less than the set parameter, switch to the left turn position.

• After that,To ensure the stability of the program at some 45-degree or so cut corners, when the smaller value of the readings of sonar[6] and sonar[7] is less than a specific parameter, also let it turn left.

• Finally,For the program of turning the outer Angle, when the small value of the readings of Sonar[7] undergoes a sudden change and is less than a certain value in the previous second, turn right until the forward distance is less than a specific value.

Step2: When there is nothing nearby, it should move straight forward. As soon as it reaches an obstacle in front, it follows the boundary of the obstacle, keeping the right side of the robot between 0.3 and 0.5 meters from the obstacle. To better determine the judgment status of the car, we have the car continuously output the currently executing program

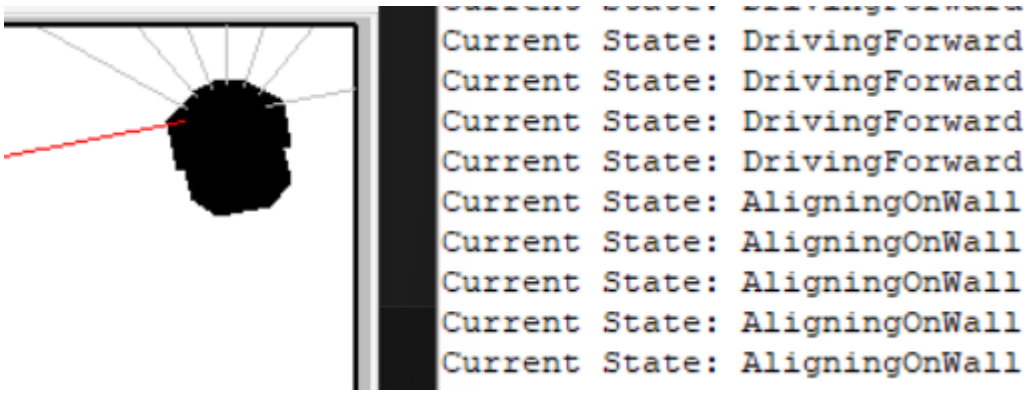

Figure 4 current determination status

General design thought

The primary navigation strategy for the mobile robot is predicated on a right-hand wall-following algorithm. To implement this behavior, a finite state machine comprising three distinct states was designed: "DrivingForward","AligningOnWall", and "TurningOuterCorner". The system initializes in the "DrivingForward" state. A state transition to "AligningOnWall" is triggered when the proximity to the wall, measured by the robot's front-right sensors, falls below a predefined threshold. In the "AligningOnWall" state, the robot executes a zero-radius rotation to achieve an orientation perpendicular to the wall surface, whereupon it immediately transitions back to the "DrivingForward" state. Furthermore, an abrupt change in the data from the right-hand sonar sensor (Sonar[7]) is interpreted as the detection of an external corner. This event triggers a transition to the "TurningOuterCorner" state, in which the robot executes the corresponding cornering maneuver.

Preparamatory parameters:

```
1. def __init__(self):
2.         """
3.         define some primary Preparamatory parameters
4.         """
5.         self.startState = 'DrivingForward'
6.         self.last_right_distance = float('inf')
7.         self.align_delay_counter = 0
8.         self.last_driving_right_distance = float('inf')
9.
10.    def getNextValues(self, state, inp):
11.        """
12.        This function runs at every step
13.        """
14.        sonars = inp.sonars
15.        front_distance = min(sonars[3], sonars[4])
16.        right_distance_far = sonars[7]
17.        right_distance_near = sonars[6]
18.        left_distance_far = sonars[0]
19.
20.        fvel = 0.0
21.        rvel = 0.0
22.        next_state = state
23.
```

Three primary states

• Driving Forward:

The DrivingForward state incorporates two principal state transition conditions and one internal corrective behavior. Firstly, a transition to the AligningOnWall state is triggered under either of two circumstances: if the lateral distance to the right-hand wall falls below a threshold of 0.25 units, or if the forward proximity measurement is less than 0.35 units. Secondly, a transition to the TurningOuterCorner state occurs upon detection of a positive step change in the right-side sensor reading that exceeds 0.1 units, which is indicative of an external corner. In addition to these state transitions, a lateral distance regulation mechanism is active within this state. To prevent the robot from deviating excessively from the wall, a corrective maneuver towards the wall is executed whenever the right-side distance surpasses 0.4 units.

DrivingForward state code:

```
1. if state == 'DrivingForward':
2.             change_in_right_distance = right_distance_far - self.last_driving_right_distance
3.
4.             if front_distance < 0.35 or min(sonars[6],sonars[7]) < 0.25:
5.                 next_state = 'AligningOnWall'
6.                 self.last_right_distance = float('inf')
7.                 self.align_delay_counter = 0
8.
9.             elif change_in_right_distance > 0.1 and \
10.                     self.last_driving_right_distance < 0.7:
11.                 next_state = 'TurningOuterCorner'
```

```
12.
13.             else:
14.                 if sonars[7] > 0.4:
15.                     fvel = 0.1
16.                     rvel = -0.3 * 3
17.                 fvel = 0.3
18.                 self.last_driving_right_distance = right_distance_far
```

- ***Aligning On Wall:***

The logic to achieve a state parallel to the wall involves real-time distance monitoring. The robot continuously samples the distance from its right-hand sensor. The moment of minimum distance is identified by comparing the current sensor reading with the previous one; this point is captured when the current reading becomes greater than the last. Immediately following the detection of this inflection point, a programmed delay is applied to make the subsequent turn smoother and less abrupt.

AligningOnWall state code:

```
1. elif state == 'AligningOnWall':
2.             if self.align_delay_counter > 0:
3.                 self.align_delay_counter -= 1
4.                 print(self.align_delay_counter)
5.                 next_state = 'AligningOnWall'
6.                 fvel = 0.0
7.                 rvel = 0.3
8.
9.                 if self.align_delay_counter == 0:
10.                    print("Timer Finished")
```

```
                    next_state = 'DrivingForward'

            else:
                current_right_distance = right_distance_far
                if current_right_distance > self.last_right_distance and
self.last_right_distance != float('inf'):
                    self.align_delay_counter = 2
                    print("Timer Started")
                    next_state = 'AligningOnWall'
                    fvel = 0.0
                    rvel = 0.3
                else:
                    next_state = 'AligningOnWall'
                    fvel = 0.0
                    rvel = 0.3
                    self.last_right_distance = current_right_distance
```

- ***Turning Outer corner:***

The logic for the TurningOuterCorner state consists of a continuous right turn using differential drive locomotion. This action persists until a wall is detected by the robot's forward sensors, at which point the system transitions back to the DrivingForward state.

TurningOuterCorner state code:

```
elif state == 'TurningOuterCorner':
            if front_distance < 0.35 * 1.5:
                next_state = 'AligningOnWall'
                self.last_right_distance = float('inf')
                self.align_delay_counter = 0
```

```
            else:
                next_state = 'TurningOuterCorner'
                fvel = 0.3 * 0.5
                rvel = -0.05 * 7
```

Summary:

This experiment successfully achieved the automatic control of the intelligent car based on sonar sensors. By programming and processing sensor data, the car can cruise stably along the wall and make smooth turns at corners through logical judgment. At the same time, it can stop immediately when it detects an obstacle in front, ensuring operational safety. The entire project transformed the theoretical closed-loop control of perception, decision-making and execution into reality. Not only did it successfully achieve the predetermined goals, but it also made us deeply realize the importance of handling sensor noise and optimizing control algorithms to cope with complex environments in actual engineering. This marks a substantial progress in our practical ability in the fields of embedded systems and robot control.