山东大学

崇新学堂

2024－2025 学年第一学期

实 验 报 告

课程名称: EECS

实验名称: Designlab02

专 业 班 级 崇新 23

学 生 姓 名 于静明 程侃 张子诺

实 验 时 间 2024/10/11

Goals

• Experiment with state machines controlling real machines
• Investigate real-world distance sensors on 6.01 robots: sonars
• Build and demonstrate a state machine to make the robot do a task: following a boundary

Step1 Materials

1. Lab laptops
2. Python 2.6.6 and the IDLE environment
3. Model car

Step2 Simple Brains

1.Run soar , choose tutorial.py and smBrain.py,we can see like this:

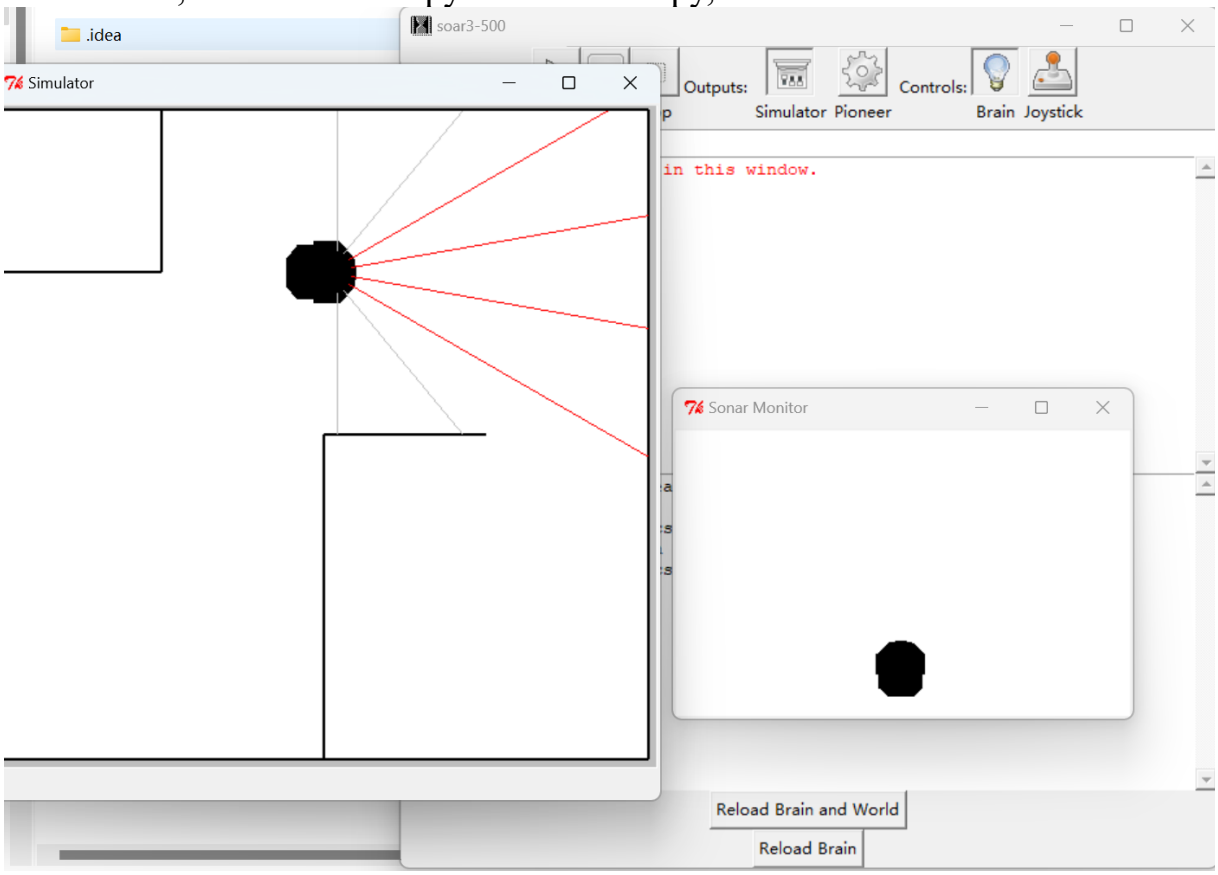

Fig 1 simulate environment

2.Run the brain by clicking the Start and the Stop buttons

Step3 Sonars

1.Modify the brain so that it sets both velocities to 0, and uncomment the line

print inp.sonars[3]

Reload the brain and run it. It will print the value of inp.sonars[3], which is the reading from one of the forward-facing sonar sensors.

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

Fig 2 code

2.the result of our code just liked this:

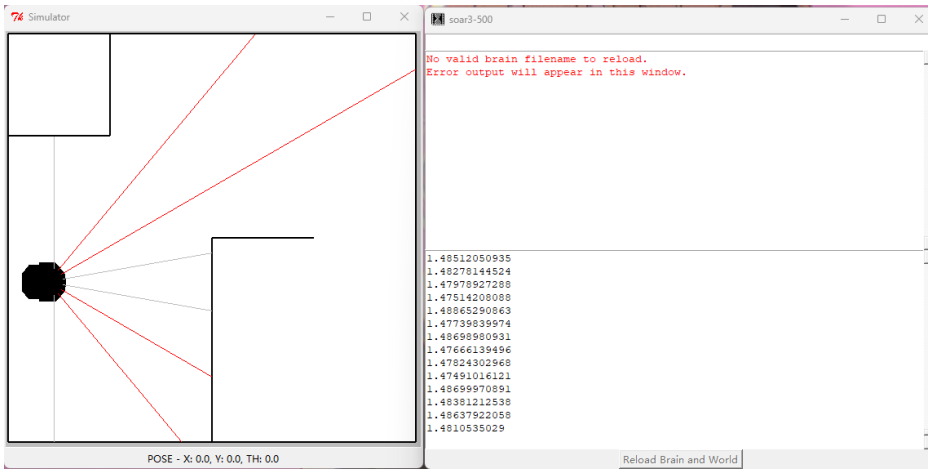

Fig 3 find the bound

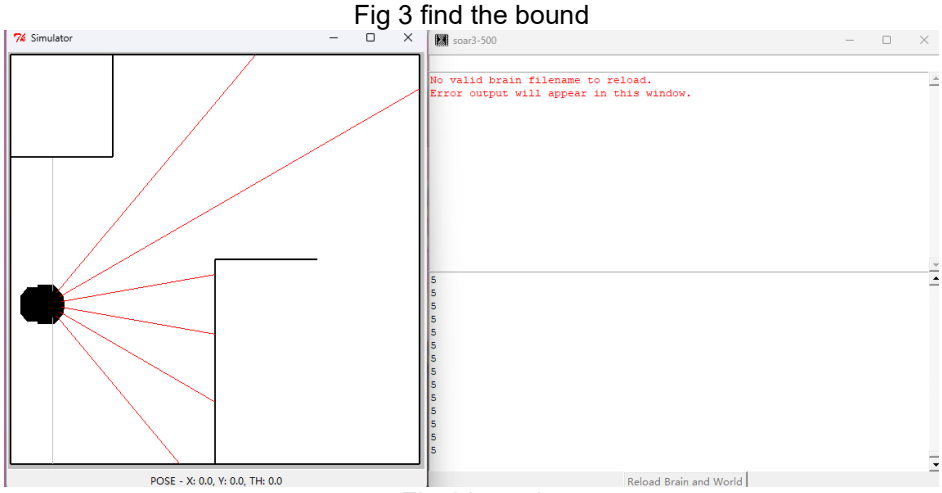

Fig 4 bound

As shown in Figures 3 and 4, when the distance is approximately within 1.5m, sonar can accurately measure the distance. However, when the distance is greater than 1.5m, sonar cannot accurately measure the distance and the return value is 5m.

3.Now, set the sonarMonitor argument to the RobotGraphics constructor to be True.

Reload the brain and run it. This will bring up a window that shows all the sonar readings graphically. The length of the beam corresponds to the reading; red beams correspond to "no valid measurement", As shown in the figure5.

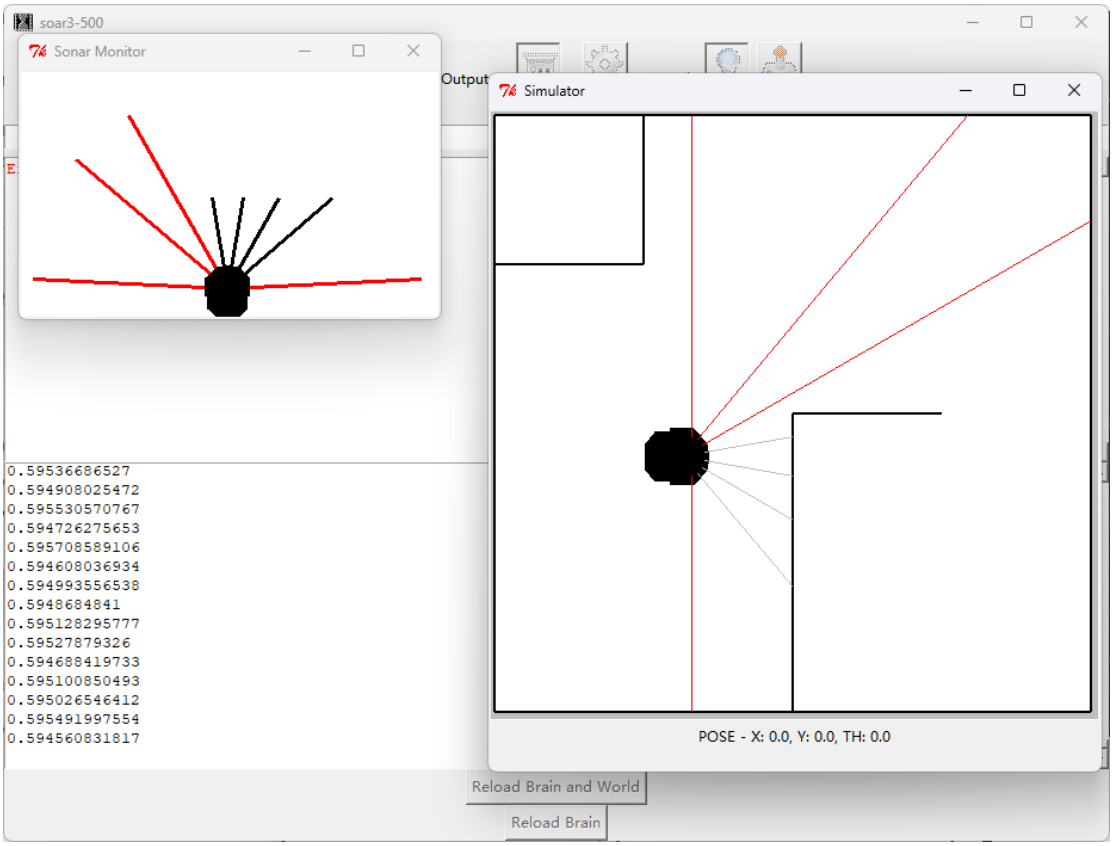

Fig 5 behavior of the sonars

Step4 Boundaries

In this part of the experiment, we were tasked with building a state machine that allows the robot to follow a boundary. Specifically, the robot should move straight forward when there are no obstacles nearby and, upon detecting an obstacle, follow the boundary by keeping the right side of the robot between 0.3 and 0.5 meters from it. The state-transition diagram is shown in **Figure 6**.Below is the explanation of the modified code that I implemented to achieve this behavior:

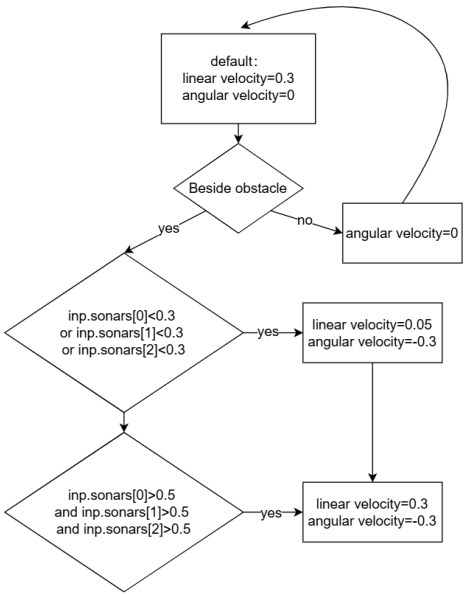

Fig 6 explanation of our code

1.Code Overview

The state machine in the code consists of a class `MySMClass` that inherits from `sm.SM`. The `getNextValues` method controls the robot's movement based on sonar sensor readings. The key logic revolves around the sonar readings on both sides of the robot, which help determine whether it should adjust its rotation or forward velocity to follow the boundary.

```
class MySMClass(sm.SM):
  def __init__(self):
    self.statement = "No obstacle"
    print("__init__()")

  def getNextValues(self, state, inp):
    f = 0.3  # Forward velocity
    r = 0    # Rotational velocity

    # If the front left, middle left, or far left sonar detects an obstacle closer than 0.3 meters
    if inp.sonars[0] < 0.3 or inp.sonars[1] < 0.3 or inp.sonars[2] < 0.3:
      r = -0.3  # Rotate counterclockwise (to move away from the obstacle)
      f = 0.05  # Slow down the forward motion
      print("close")
    # If all three left sonars detect no obstacles within 0.5 meters
    if inp.sonars[0] > 0.5 and inp.sonars[1] > 0.5 and inp.sonars[2] > 0.5:
      r = 0.3  # Rotate clockwise (to adjust towards the boundary)
      f = 0.3  # Move forward at regular speed
      print("far")
    # If the right sonar sensors detect an obstacle too close (less than 0.05 meters)
    if inp.sonars[5] < 0.05 or inp.sonars[6] < 0.05 or inp.sonars[7] < 0.05:
      r = -0.3  # Rotate counterclockwise to avoid the obstacle
      f = -0.1  # Move backward slightly
    # If no obstacle is detected, evaluate the overall situation
    if self.statement == "No obstacle":
      r = 0  # No rotation
      flag = 1  # Flag to check if all sonar readings are clear
      # Loop through all sonar readings to verify no obstacle is within 0.3 meters
      for i in range(0, 8):
        print(inp.sonars[i])  # Print sonar values for debugging
        flag = flag * (inp.sonars[i] > 0.3)
      if flag:  # If all sonar readings indicate no obstacles
        print("no obstacle")
      else:  # If an obstacle is detected, update the state
        self.statement = "Beside obstacle"
        print(self.statement)
    # Print and adjust final velocities
    print(f, r)
    f = f * 1
    r = r * 6  # Amplify rotational velocity
    return (state, io.Action(fvel=f, rvel=r))
```

Fig 7 backbone of our work

2.Key Modifications and Logic:

a. **Obstacle Detection:**

The robot uses sonar readings to detect obstacles in front and on its sides.

If the left-front sonar sensors (sonars 0, 1, 2) detect an obstacle within 0.3 meters, the robot slows down and rotates counterclockwise to avoid collision.

Conversely, if no obstacles are detected within 0.5 meters on the left, the robot rotates clockwise to re-align itself closer to the boundary.

b. **Too Close on the Right:**

If the sonar sensors on the right side (sonars 5, 6, 7) detect an obstacle too close (less than 0.05 meters), the robot rotates counterclockwise and slightly moves backward to avoid collision.

c. **State Handling for "No Obstacle":**

When there is no obstacle detected, the robot continues to move straight ahead. The loop checks all sonar sensors to confirm if it is truly clear of obstacles.

If all sensors indicate no obstacles (readings greater than 0.3 meters), the state remains "No obstacle."

If an obstacle is detected, the state changes to "Beside obstacle," prompting the robot to adjust its movement accordingly.

d. **Velocity Adjustments:**

The forward velocity `f` is kept at 0.3 by default but is adjusted based on proximity to obstacles.

The rotational velocity `r` is scaled by 6 to amplify its effect on the robot's turning behavior.

3.Result:

With these modifications, the robot in the simulator is able to:

Move forward when no obstacle is detected.

Adjust its trajectory to follow the boundary while maintaining a safe distance, as shown in Figure 8,9.

Handle sharp turns and respond to obstacles effectively using the sonar sensor inputs, as illustrated in the simulation.

Additionally, we performed tests using the actual robot. The robot successfully completed the task of following the boundary closely, adhering to the safe distance from obstacles during real-world tests. The real robot's performance mirrored the behavior observed in the simulator, handling both straight and corner boundaries effectively.

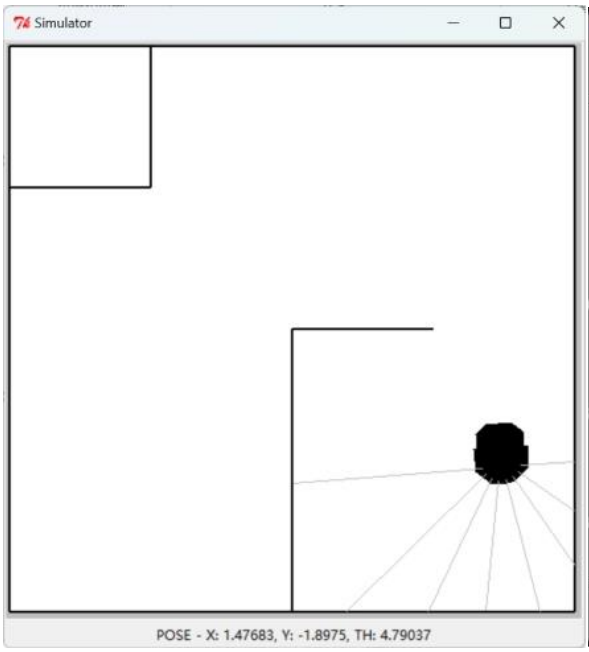

Figure 8 map

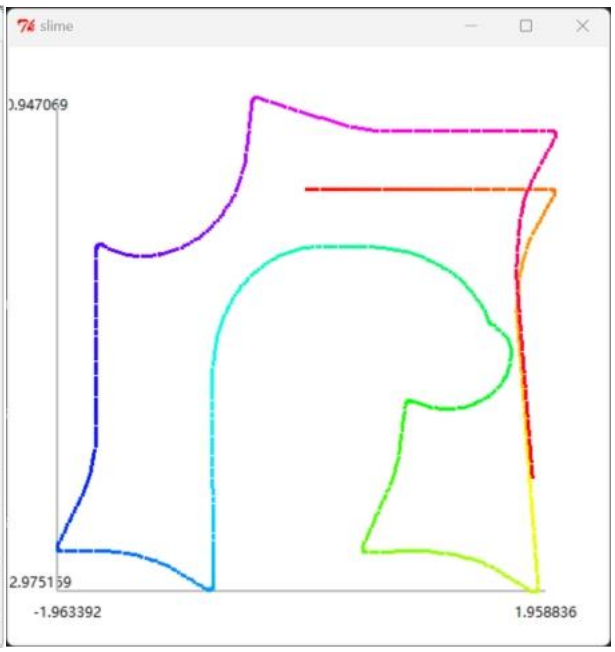

Figure 9 track

Step5 Real car test

Through the pioneer button to burn our program into the car, fortunately, we achieved success the first time, did not go through the process of tuning parameters, very lucky

Summary

In this experiment, our group through cooperation realized the exploration of sonar properties, the fixed distance tracking of the car to the obstacle and the obstacle avoidance of the car