**崇新学堂**

**2024－2025学年第一学期**

实 验 报 告

课程名称：电子信息工程导论实验

实验名称：Controlling Robots

专 业 班 级 崇新学堂

学 生 姓 名 胡君安、陈焕斌、黄颢

实 验 时 间 2024年10月10日

**Step1 Run a brain in the simulator**

Open the Soar interface,as shown in Figure 1.

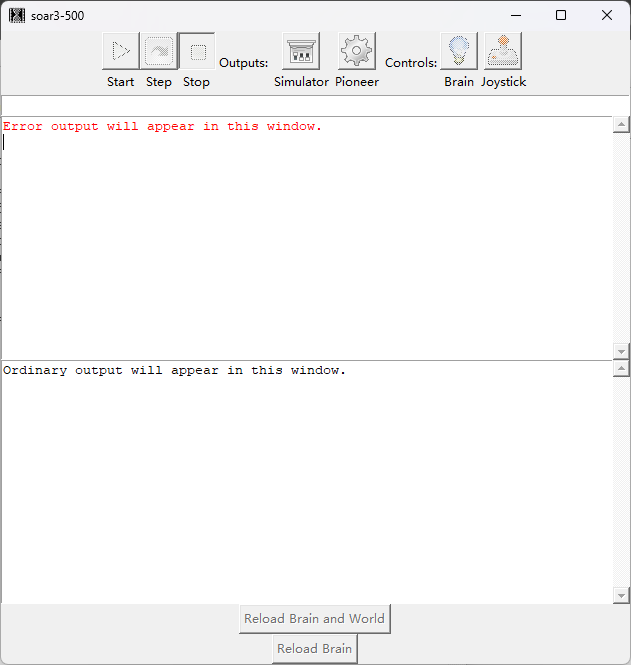


Figure Soar Interface

Click soar’s Simulator button and double-click tutorial.py.This loads a specific virtual

world into our robot simulator.

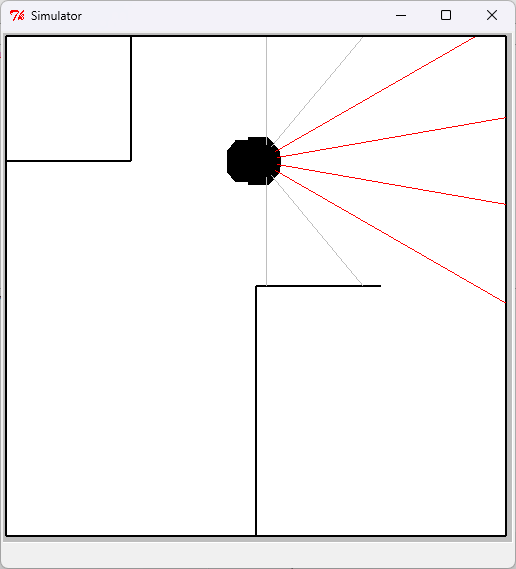


Figure Simulator Interface

Click soar’s Brain button, navigate to …/6.01/designLab02/smBrain.py, and

click Open.Then we have successfully loaded the Brain.

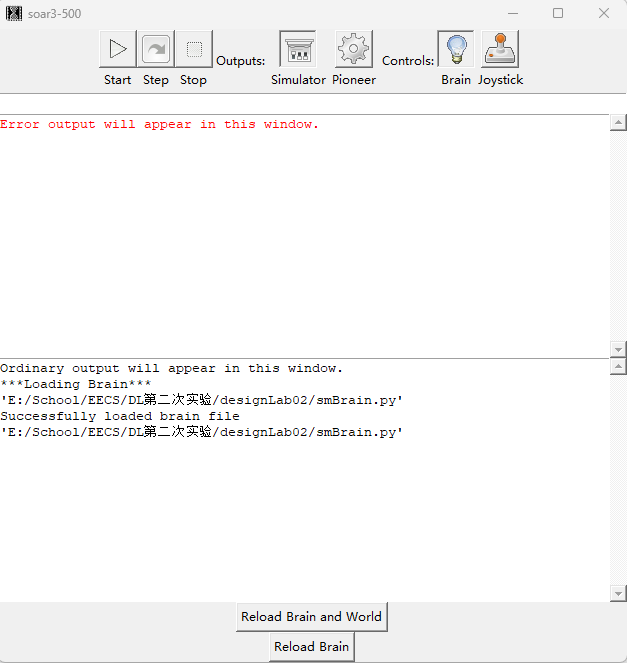


Figure Loading the Brain

Click soar’s Start button, and let the robot run for a little while.Then,click the Stop button.Notice the graph that was produced.

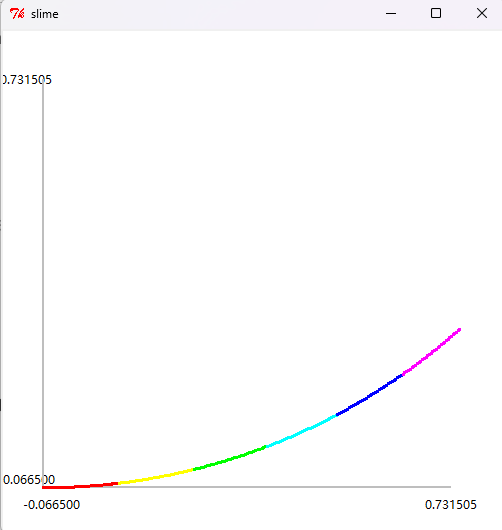


Figure “Slime Trail”

**Step2 Modify the brain and run it.**

Open smBrain.py through IDLE.

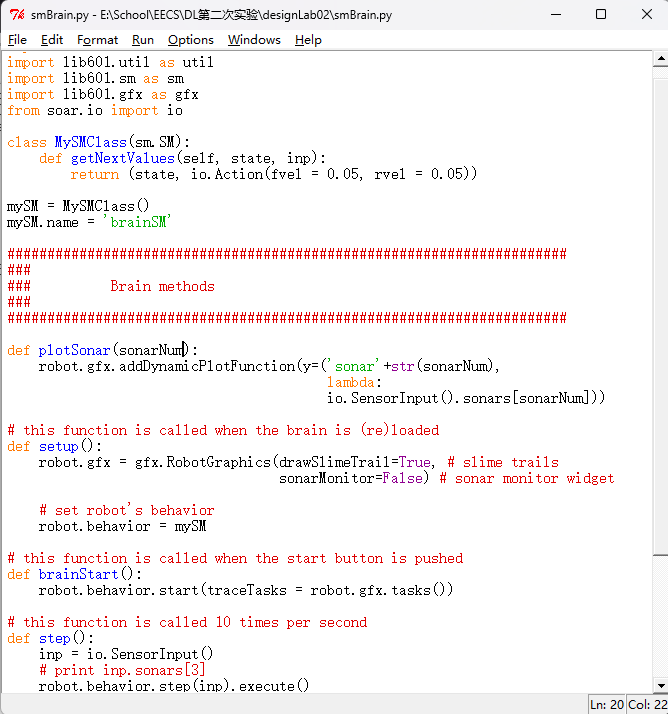


Figure smBrain.py Interface

We understand that the speed is determined by fvel and rvel. Among them, fvel is forward velocity of the robot and rvel is rotational velocity of the robot.

Set fvel to 0 so that it makes the simulated robot rotate in place.

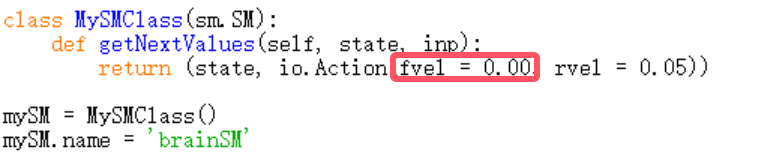


Figure Set fvel

Then we can run it on a robot to test.

**Checkoff 1. Wk.2.2.1: Explain to a staff member the results of your experiments with the sonars.**

Uncomment the line *print inp.sonars[3].*

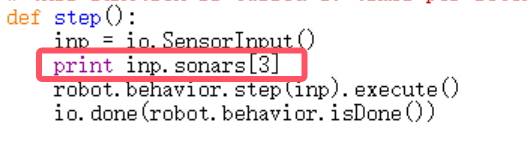


Figure Uncomment the line

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.

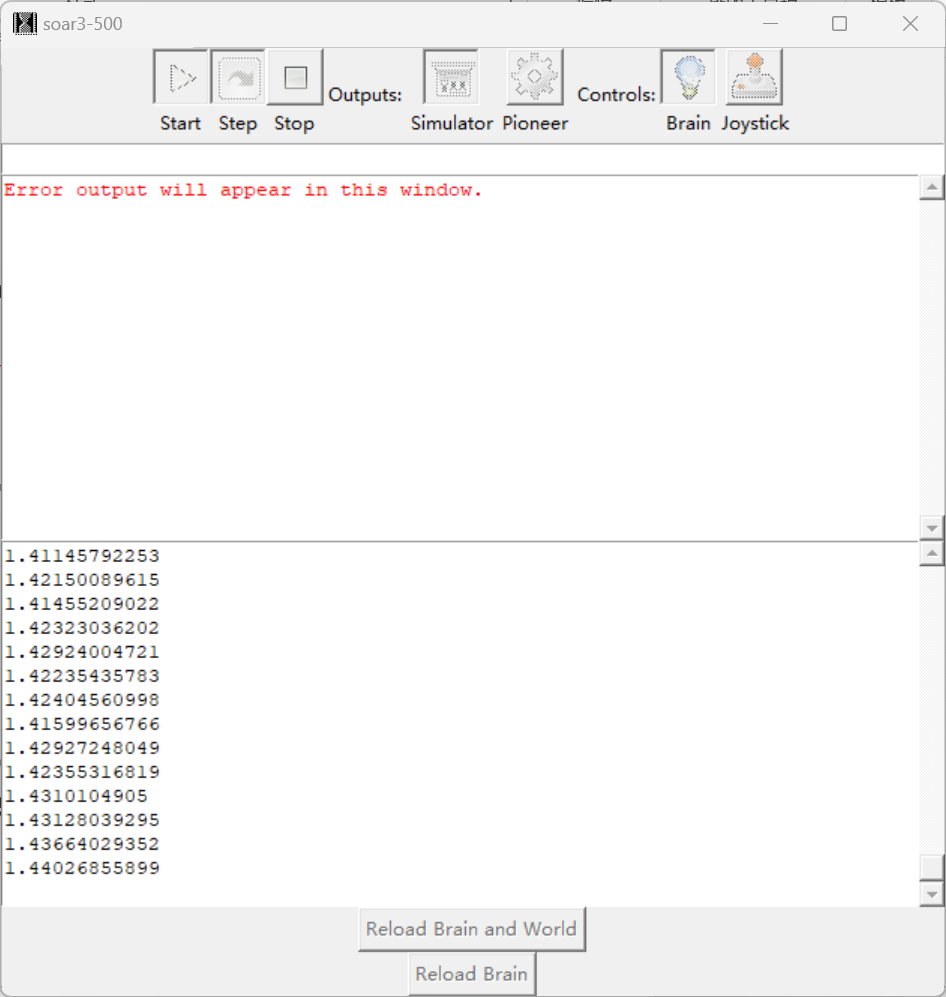


Figure Sonar feedback

We can know that:Capable of obtaining valid readings from a maximum distance of 1.5;If the distance is greater than 1.5, you will continue to receive 5,which is because the distance is too far to obtain the value.If something is too close to the car, the car will turn red and the reading will fluctuate.

Changing the angle between the sonar and the surface that it is pointed toward can affect the readings.This behavior depend on the material of the surface.The bubble wrap can accurately transmit the distance of the radar, and the car can operate normally according to the program.However,if a smooth foam core is used, the car cannot execute the steering logic and will walk straight and hit the wall.

Set the sonarMonitor argument to the RobotGraphics constructor to be True.

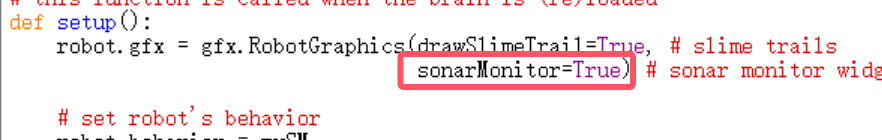


Figure SonarMonitor to be True

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”. Test that all sonars are working by blocking each one in turn.Through testing,we understand that all the sonars are working.

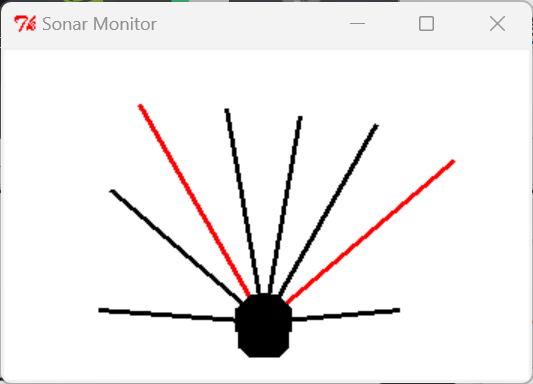


Figure Sonar Monitor

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

We have verified with TA during the experimental process,the relevant brain code is as follows:

import math

import lib601.util as util

import lib601.sm as sm

import lib601.gfx as gfx

from soar.io import io

class MySMClass(sm.SM):

def getNextValues(self, state, inp):

if inp.sonars[3] < 0.55:

return (state, io.Action(fvel = 0.00, rvel = 0.00))

else:

return (state, io.Action(fvel = 0.1, rvel = 0.0))

mySM = MySMClass()

mySM.name = 'brainSM'

######################################################################

###

### Brain methods

###

######################################################################

def plotSonar(sonarNum):

robot.gfx.addDynamicPlotFunction(y=('sonar'+str(sonarNum),

lambda:

io.SensorInput().sonars[sonarNum]))

# this function is called when the brain is (re)loaded

def setup():

robot.gfx = gfx.RobotGraphics(drawSlimeTrail=True, # slime trails

sonarMonitor=True) # sonar monitor widget

# set robot's behavior

robot.behavior = mySM

# this function is called when the start button is pushed

def brainStart():

robot.behavior.start(traceTasks = robot.gfx.tasks())

# this function is called 10 times per second

def step():

inp = io.SensorInput()

print inp.sonars[3]

robot.behavior.step(inp).execute()

io.done(robot.behavior.isDone())

# called when the stop button is pushed

def brainStop():

pass

# called when brain or world is reloaded (before setup)

def shutdown():

pass

**Checkoff 3** **Show your state-transition diagram to a staff member. Make clear what the conditions on state transitions are, and what actions are associated with each state.**

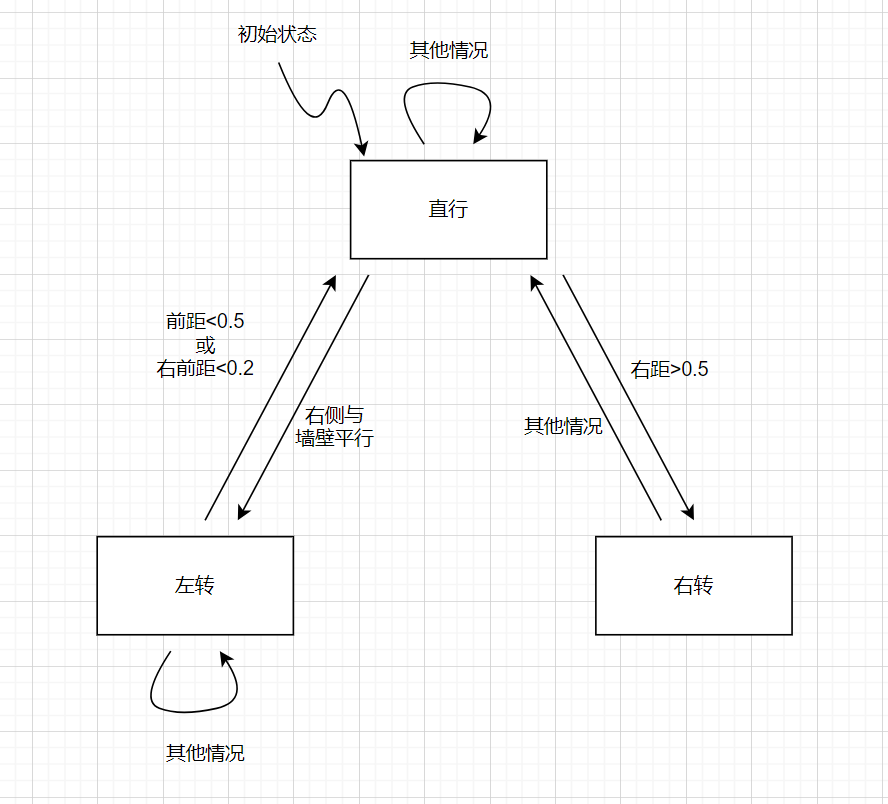


Figure 11 State Machine

**Checkoff 4 Wk.2.2.4: Demonstrate your boundary follower to a staff member. Explain why it behaves the way it does.**

We have verified with TA during the experimental process,the relevant brain code is as follows:

import math

import lib601.util as util

import lib601.sm as sm

import lib601.gfx as gfx

from soar.io import io

class MySMClass(sm.SM):

def \_\_init\_\_(self):

self.startState = 'forward'

def getNextValues(self, state, inp):

front\_distance = min(inp.sonars[3:4])

distance5 = inp.sonars[5]

distance6 = inp.sonars[6]

distance7 = inp.sonars[7]

if state == 'forward':

if front\_distance<0.5 or min(distance5,distance6,distance7)<0.2:

return ('turnleft', io.Action(fvel = 0.1, rvel = 0.8))

elif distance7>0.5:

return ('turnright', io.Action(fvel = 0.10, rvel = -0.5))

else:

return ('forward', io.Action(fvel = 0.1, rvel = 0))

elif state == 'turnright':

return ('forward', io.Action(fvel = 0.1, rvel = 0))

elif state == 'turnleft':

#180/7=25.7

#cos(25.7)=0.9,cos(25.7\*3)=0.22

if (distance7<0.9\*distance6 and distance7<distance6) or (distance7<0.6\*distance5 and distance7<distance5) or distance7>0.8:

return ('forward', io.Action(fvel = 0.1, rvel = 0))

else:

return ('turnleft', io.Action(fvel = 0.1, rvel = 0.3))

MySMClass.getNextValues

mySM = MySMClass()

mySM.name = 'brainSM'

######################################################################

###

### Brain methods

###

######################################################################

def plotSonar(sonarNum):

robot.gfx.addDynamicPlotFunction(y=('sonar'+str(sonarNum),

lambda:

io.SensorInput().sonars[sonarNum]))

# this function is called when the brain is (re)loaded

def setup():

robot.gfx = gfx.RobotGraphics(drawSlimeTrail=True, # slime trails

sonarMonitor=True) # sonar monitor widget

# set robot's behavior

robot.behavior = mySM

# this function is called when the start button is pushed

def brainStart():

robot.behavior.start(traceTasks = robot.gfx.tasks())

# this function is called 10 times per second

def step():

inp = io.SensorInput()

print inp.sonars[3]

robot.behavior.step(inp).execute()

io.done(robot.behavior.isDone())

# called when the stop button is pushed

def brainStop():

pass

# called when brain or world is reloaded (before setup)

def shutdown():

pass

Explanation of logic:

We cleverly used mathematical geometric relationships to determine the logic of transitioning from a left turn state to a forward state. We know that for adjacent sensors, the angle between them should be about 25.7 °. When the car completes a left turn and is about to enter a forward state, the right side of the car should be parallel to the wall. Distance7, distance6 form a right angled triangle with the wall.

At this point, cos25.7 ° = distance7/distance6.

This is a **highlight** of our experimental code.

The following figure shows the simulation results of the code (almost perfect).

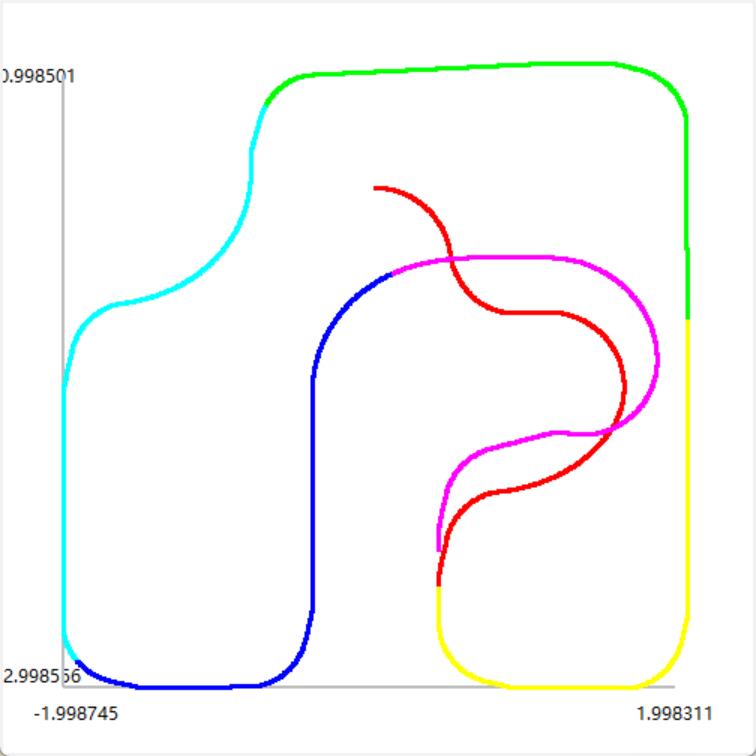


Figure 12 Path simulation

**Summary**

The experiment focused on using state machines to control a robot, specifically in navigating its environment and interacting with objects via sonar sensors. The primary goal was to implement a state machine that allows the robot to follow a boundary while maintaining a specific distance from obstacles. The experiment involved multiple stages, including simulating the robot's behavior in a virtual environment and then applying the developed brain to a real robot.

In the first phase, we used the smBrain.py program to control the robot in a simulated world, refining it to achieve specific tasks such as rotating in place. In the second phase, we transitioned to real-world robotics by applying the same state machine and focusing on the behavior of sonar sensors. We investigated the accuracy and limitations of the robot's sonar sensors, experimented with distance-keeping, and ultimately programmed the robot to move forward and adjust its distance based on obstacles it encountered.

This experiment provided valuable hands-on experience with state machines and their practical applications in robotics. By controlling a robot and modifying its behavior in response to sensory data, we gained insights into the complexities of real-world navigation systems. Testing the robot’s ability to follow a boundary helped to understand the importance of sensor accuracy and response time in dynamic environments. Additionally, working in both simulated and physical environments illustrated the differences and challenges posed by real-world conditions compared to simulations.

We met plenty of problems, for example, the robot initially struggled to maintain a consistent distance from obstacles when they moved or changed position. Regarding this issue, We modified the robot’s state machine to refine the distance-keeping behavior by fine-tuning the velocity parameters and testing in both the simulator and the real world. Debugging the logic in the getNextValues method resolved these inconsistencies, allowing the robot to respond more accurately to varying distances. The most troublesome problem was that the robot had difficulties navigating sharp corners, especially turning left. To solve this problem, we added the rvel when turning left. However, the problem that came with it was that the right turn couldn’t be set too fast, otherwise the robot's sonar would not be able to correctly detect the turn and the robot will rotate a large angle when adjusting the distance from the right wall. Therefore, we reduced the rvel when turning right, but this cause the robot unable to finish the first right turn of the second lap. Because for right turns, we did not have a state of continuing to turn right after turning right, which would make the robot far from the wall, leading to incorrect determination of states. We used oblique sonar to make complex judgments, which enabled the robot to complete the journey normally and continue on the second lap.