山东大学

崇新学堂

2022－2023 学年第一学期

实 验 报 告

课程名称： EECS

实验名称： Designlab3

专　业　班　级	崇新 21
学　生　姓　名	张原 刘浩 施政 池弋
实　验　时　间	2022.10.14

Step1 make the composite machine:

According to the experimental requirements and figure 1, we created the state machine:

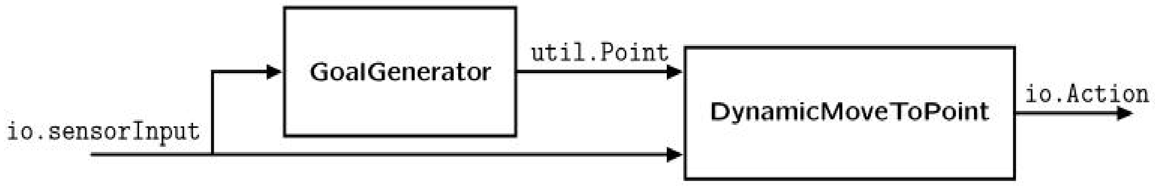
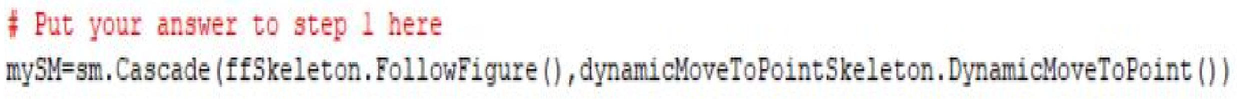


Figure 1 composite machine

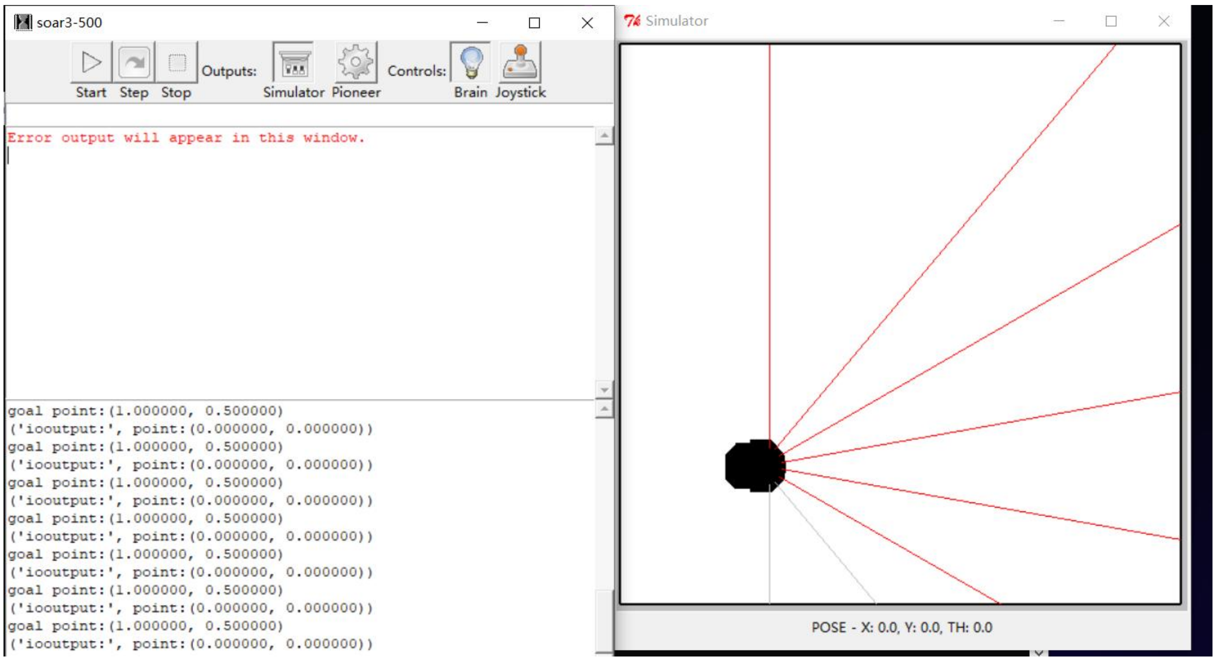
We edited the GoalGenerator in the ffSkeleton.py file, and creating an instance of the state machine calling FollowFigure. It would output the fixed point——(1.0,0.5). Meanwhile,we edited the DynamicMoveToPoint of state machine in thedynamicMoveToPointSkeleton.py file, but let it do nothing. And we type my code for constructing this composite machine into moveBrainSkeleton.py,as shown in the figure 2:



```
# Put your answer to step 1 here
mySM=sm.Cascade(ffSkeleton.FollowFigure(),dynamicMoveToPointSkeleton.DynamicMoveToPoint())
```

Figure 2 my code

The final result is shown in Figure 3:



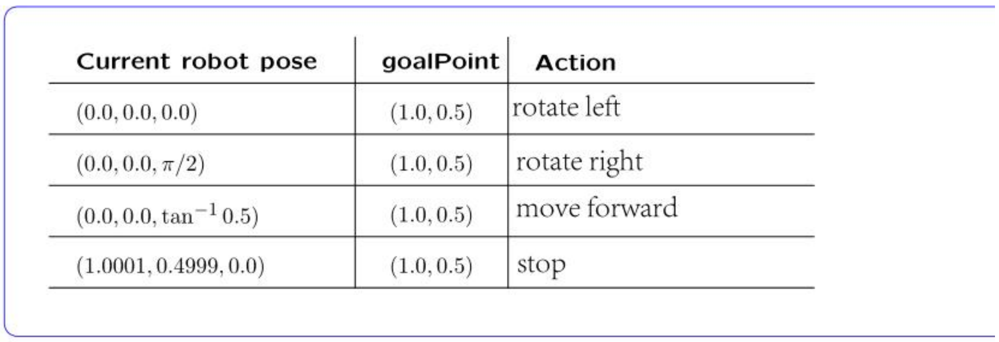
```
Error output will appear in this window.
```

```
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
goal point:(1.000000, 0.500000)
('iooutput:', point:(0.000000, 0.000000))
```

Figure 3 final result

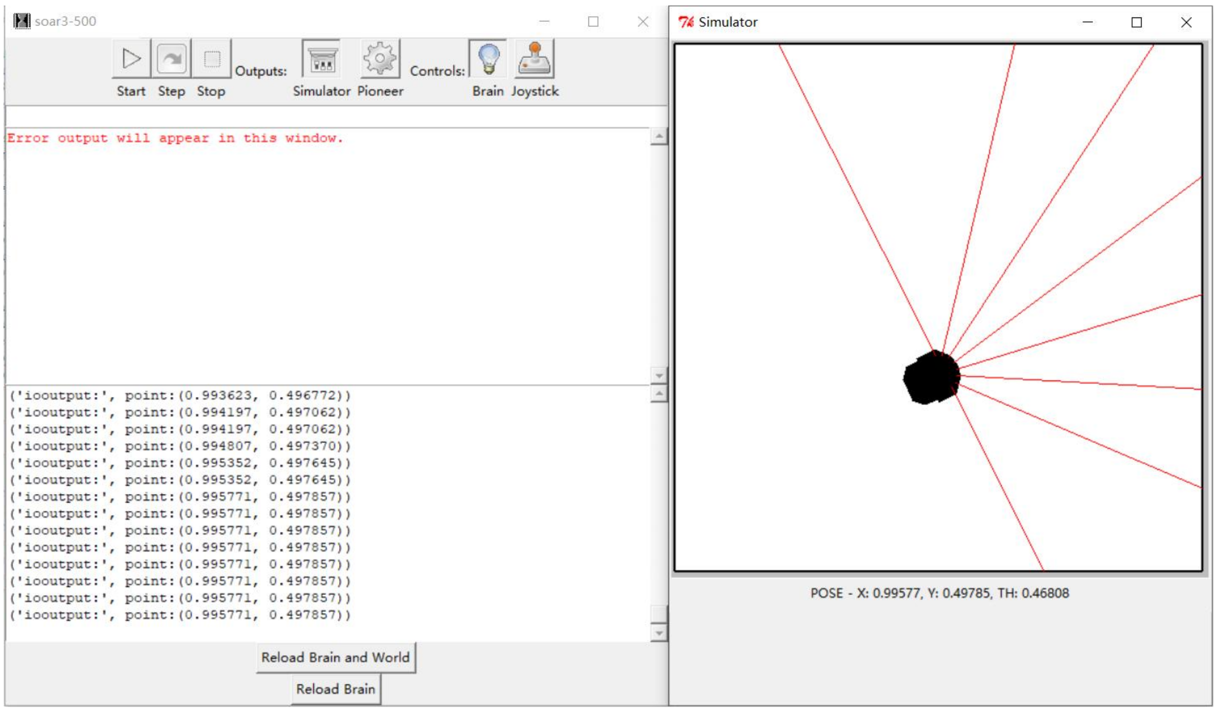
Step2 Determine the action

Input a tuple of the current position and the target point, by measuring the gap between the two points to determine the output of what action.



Current robot pose	goalPoint	Action
$(0.0, 0.0, 0.0)$	$(1.0, 0.5)$	rotate left
$(0.0, 0.0, \pi/2)$	$(1.0, 0.5)$	rotate right
$(0.0, 0.0, \tan^{-1} 0.5)$	$(1.0, 0.5)$	move forward
$(1.0001, 0.4999, 0.0)$	$(1.0, 0.5)$	stop

Figure 4 the action of car



```
Error output will appear in this window.
```

```
('iooutput:', point:(0.993623, 0.496772))
('iooutput:', point:(0.994197, 0.497062))
('iooutput:', point:(0.994197, 0.497062))
('iooutput:', point:(0.994807, 0.497370))
('iooutput:', point:(0.995352, 0.497645))
('iooutput:', point:(0.995352, 0.497645))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
('iooutput:', point:(0.995771, 0.497857))
```

```
POSE - X: 0.99577, Y: 0.49785, TH: 0.46808
```

Figure 5 Reach target point

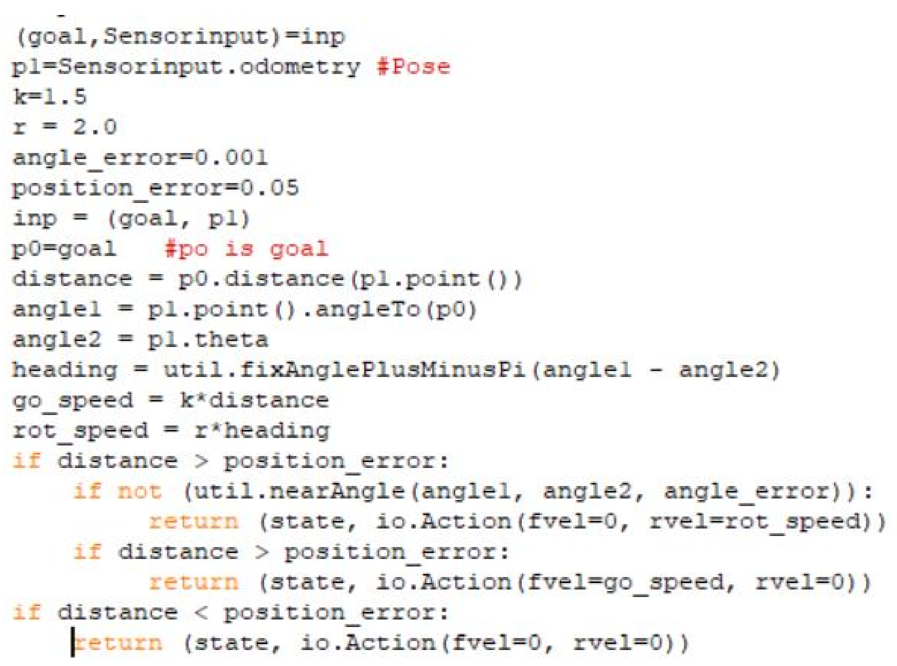
Checkoff 1. **Explain your strategy for implementing this behavior and your answers to the questions above to a staff member.**

Using p0.distance(p1) and p0.angleTo(p1), we get the distance---'distance' and Angle difference---'angle1'. Let's say the Angle of car's current position is equal to 'angle2'.

Using util.nearAngle , we can judge whether there is an Angle difference between the current position and the target point. If the Angle difference is not within the error range, the rotation action is output. The direction of rotation depends on the difference of 'angle1' and 'angle2'. If the Angle difference is within the error range, it indicates that the Angle has been adjusted, and then check the distance. If the distance is within a certain error range, it means that the car has reached the target point. When the target point is reached, it stops moving.

Step3 Write code:

The whole process code of getNextValues() is shown in Figure 6:



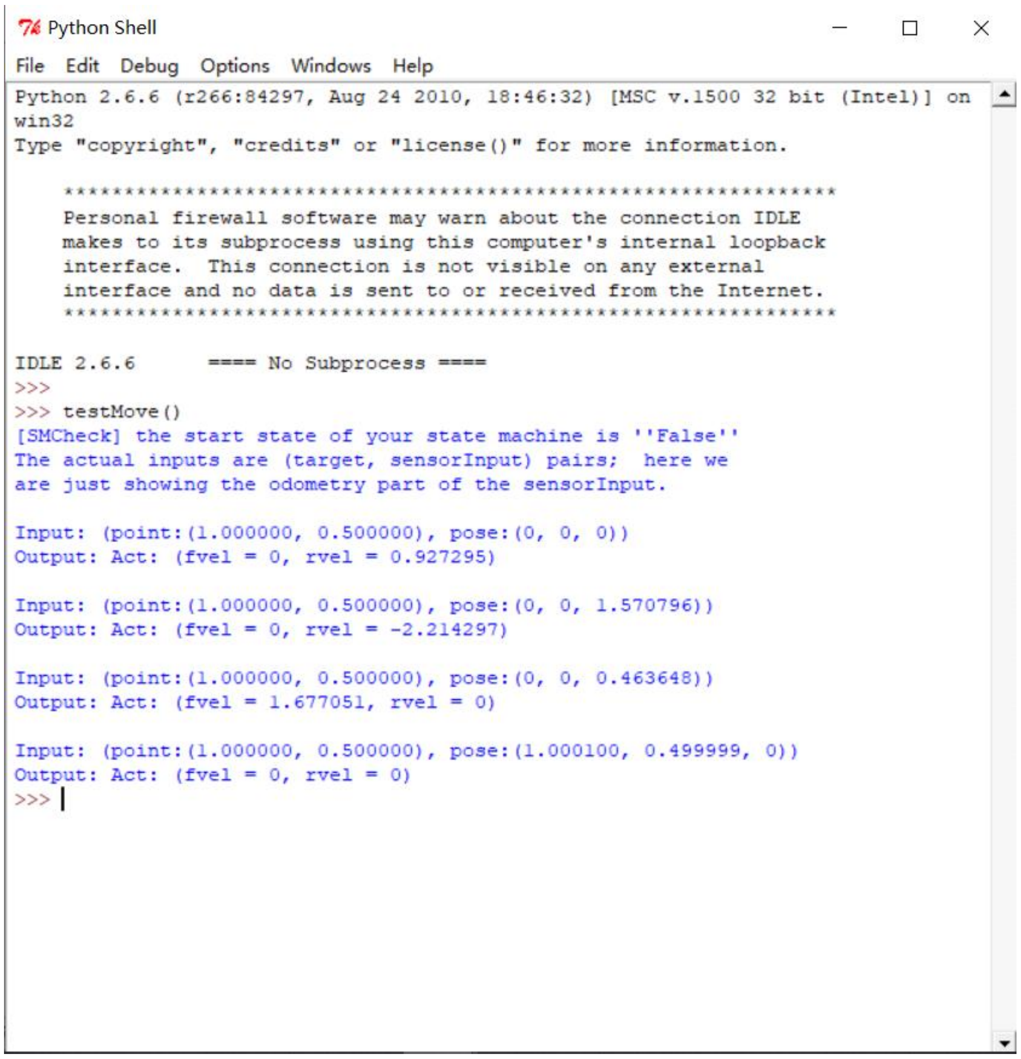
```
(goal,Sensorinput)=inp
p1=Sensorinput.odometry #Pose
k=1.5
r = 2.0
angle_error=0.001
position_error=0.05
inp = (goal, p1)
p0=goal    #po is goal
distance = p0.distance(p1.point())
angle1 = p1.point().angleTo(p0)
angle2 = p1.theta
heading = util.fixAnglePlusMinusPi(angle1 - angle2)
go_speed = k*distance
rot_speed = r*heading
if distance > position_error:
    if not (util.nearAngle(angle1, angle2, angle_error)):
         return (state, io.Action(fvel=0, rvel=rot_speed))
    if distance > position_error:
         return (state, io.Action(fvel=go_speed, rvel=0))
if distance < position_error:
    return (state, io.Action(fvel=0, rvel=0))
```

Figure 6 whole process code

Check Yourself 1. **Be sure you understand what the answers to the test cases in this file ought to be, and that your code is generating them correctly.**

When the current position does not reach the target point, the rotation is performed. When the direction is right, the car moves.When the target point is reached, the car stops moving.

The result of validation is shown in Figure 7:



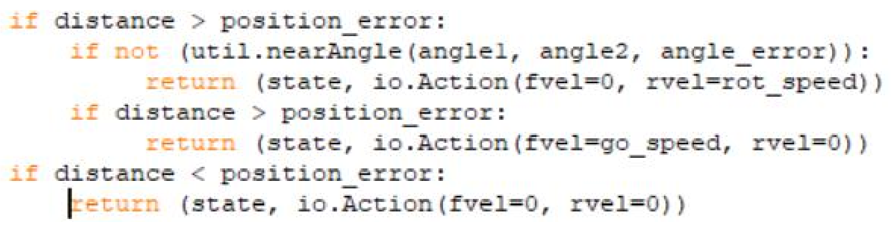
```
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.6      ==== No Subprocess ====
>>>
>>> testMove()
[SMCheck] the start state of your state machine is ''False''
The actual inputs are (target, sensorInput) pairs;  here we
are just showing the odometry part of the sensorInput.

Input: (point:(1.000000, 0.500000), pose:(0, 0, 0))
Output: Act: (fvel = 0, rvel = 0.927295)

Input: (point:(1.000000, 0.500000), pose:(0, 0, 1.570796))
Output: Act: (fvel = 0, rvel = -2.214297)

Input: (point:(1.000000, 0.500000), pose:(0, 0, 0.463648))
Output: Act: (fvel = 1.677051, rvel = 0)

Input: (point:(1.000000, 0.500000), pose:(1.000100, 0.499999, 0))
Output: Act: (fvel = 0, rvel = 0)
>>> |
```

Figure 7 the result of testmove

Check Yourself 2:**The robot always starts with odometry reading (0, 0, 0), so it ought to move to somewhere close to the point (1.0, 0.5) and stop.**

The distance is used to determine whether the position is reached. If the car is not reached, the Angle and distance are adjusted. If it is reached, the car will stop.

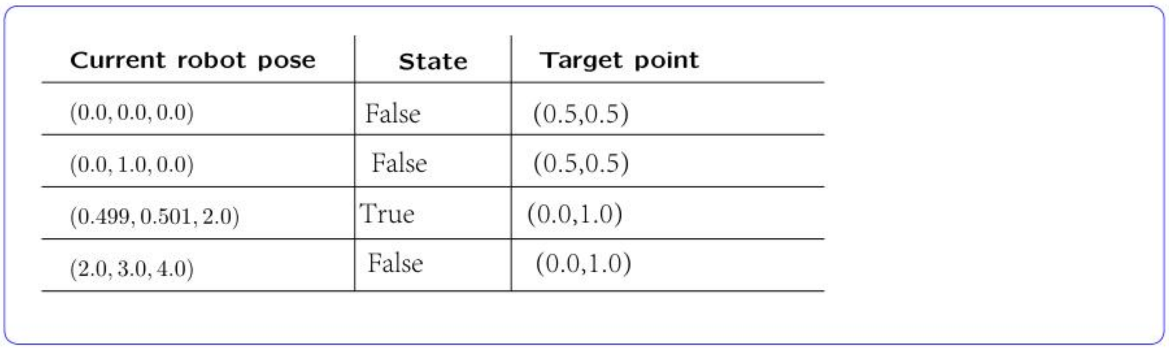
The implementation of code is as follows:



```
if distance > position_error:
    if not (util.nearAngle(angle1, angle2, angle_error)):
        return (state, io.Action(fvel=0, rvel=rot_speed))
    if distance > position_error:
        return (state, io.Action(fvel=go_speed, rvel=0))
if distance < position_error:
    return (state, io.Action(fvel=0, rvel=0))
```

Figure 8 move to point

Step4 Define our FollowFigure state machine:

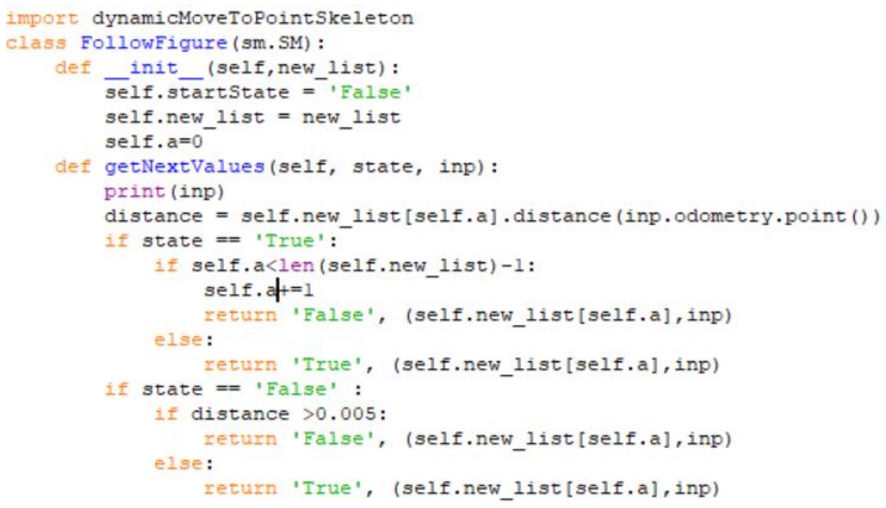


Current robot pose	State	Target point
(0.0, 0.0, 0.0)	False	(0.5,0.5)
(0.0, 1.0, 0.0)	False	(0.5,0.5)
(0.499, 0.501, 2.0)	True	(0.0,1.0)
(2.0, 3.0, 4.0)	False	(0.0,1.0)

Figure 9 the state and target point

A list of target points is inputted into the state machine, and whether the target point is reached can be determined by the distance. If the target point is reached, it returns 'True' and switches to the next point.When the last goal is reached, keep outputting this point.

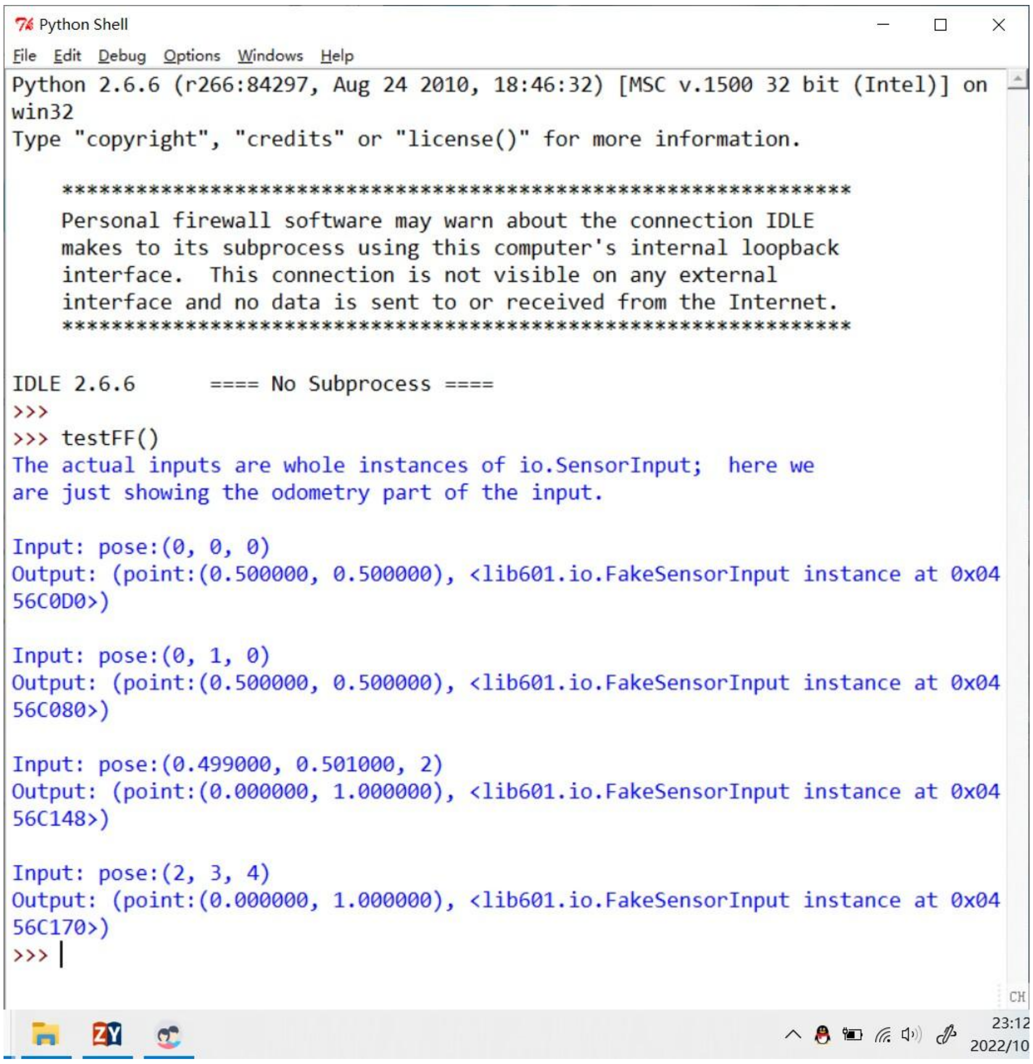
The following code:



```
import dynamicMoveToPointSkeleton
class FollowFigure(sm.SM):
    def __init__(self,new_list):
        self.startState = 'False'
        self.new_list = new_list
        self.a=0
    def getNextValues(self, state, inp):
        print(inp)
        distance = self.new_list[self.a].distance(inp.odometry.point())
        if state == 'True':
            if self.a<len(self.new_list)-1:
                self.a+=1
                return 'False', (self.new_list[self.a],inp)
            else:
                return 'True', (self.new_list[self.a],inp)
        if state == 'False' :
            if distance >0.005:
                return 'False', (self.new_list[self.a],inp)
            else:
                return 'True', (self.new_list[self.a],inp)
```

Figure 10 FollowFigure of code

Explain the answer:The first two position points do not reach (0.5, 0.5). So (0.5,0.5) was continued to output.Until the third point passes the decision, and then switches to the next point(0.0,1.0).The fourth point does not reach the target, so it outputs (0.0, 1.0) again.



```
Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.6      ==== No Subprocess ====
>>>
>>> testFF()
The actual inputs are whole instances of io.SensorInput;  here we
are just showing the odometry part of the input.

Input: pose:(0, 0, 0)
Output: (point:(0.500000, 0.500000), <lib601.io.FakeSensorInput instance at 0x04
56C0D0>)

Input: pose:(0, 1, 0)
Output: (point:(0.500000, 0.500000), <lib601.io.FakeSensorInput instance at 0x04
56C080>)

Input: pose:(0.499000, 0.501000, 2)
Output: (point:(0.000000, 1.000000), <lib601.io.FakeSensorInput instance at 0x04
56C148>)

Input: pose:(2, 3, 4)
Output: (point:(0.000000, 1.000000), <lib601.io.FakeSensorInput instance at 0x04
56C170>)
>>> |
```

Figure11 the result of testFF

Step5 and Checkoff 2. **Show the slime trail resulting from the simulated robot moving in a square or other interesting figure to a staff member. Explain why it has the shape it does. Take a**

screenshot of the slime trail and save it for your interview. Mail the code and slime trail you have so far, to your partner.

Enter the file—— secretMessage into the main program, and the slime trail is as follows:

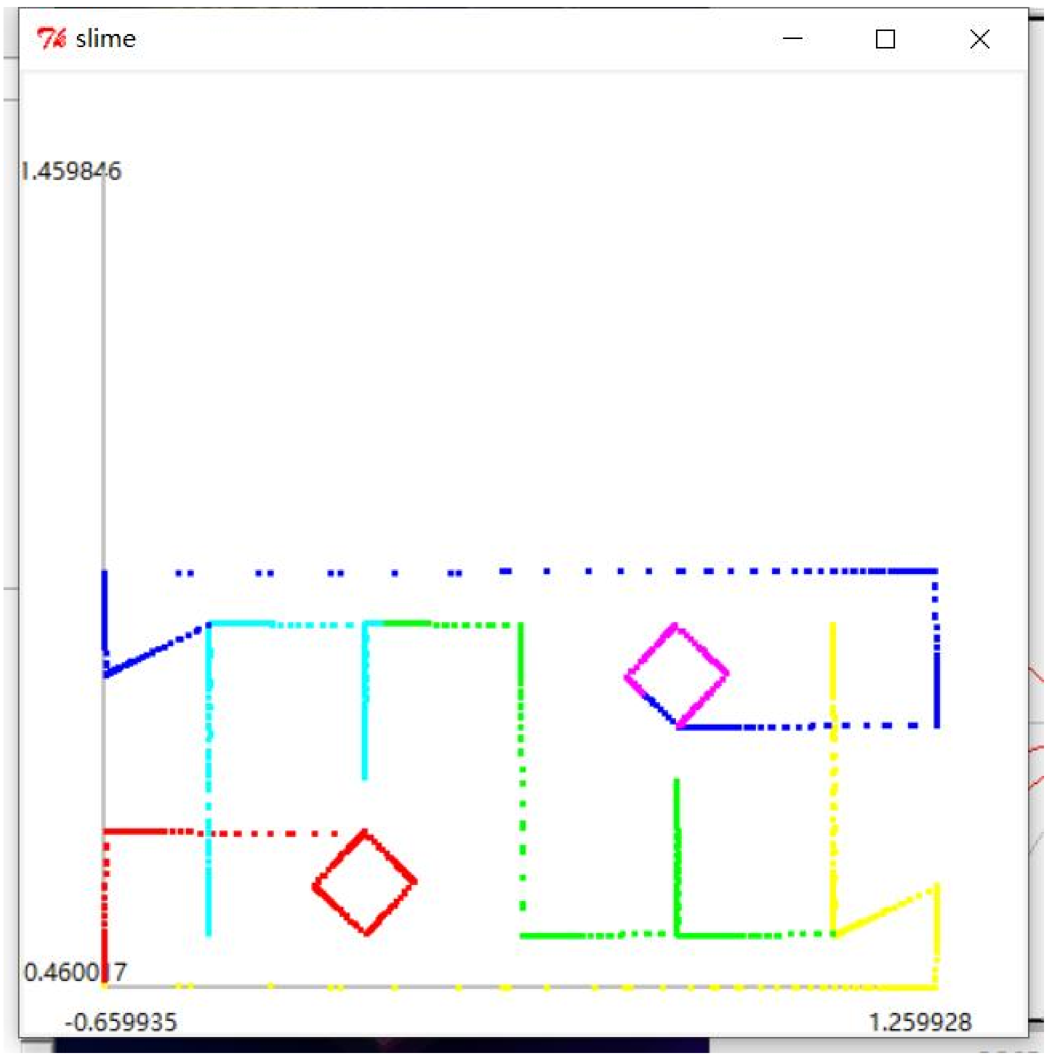


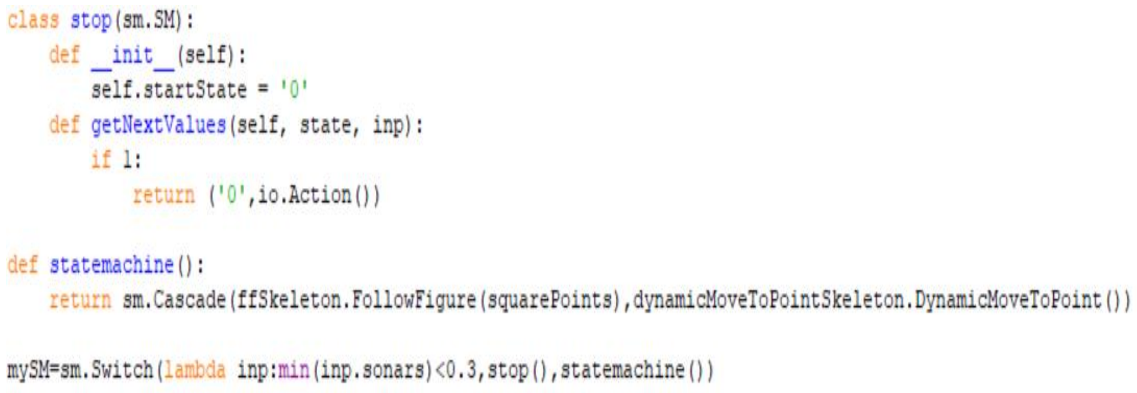
Figure 12 secretMessage

The trajectory presents an origin symmetric MIT

Step6 keeping a safe distance:

According to the experimental requirements, we use the sm.Switch to achieve the car obstacle avoidance.We define a 'stop state machine', where if any of the sonar detects a distance less than 0.3, it enters the stop state machine.The car will remain stationary.

The following code:

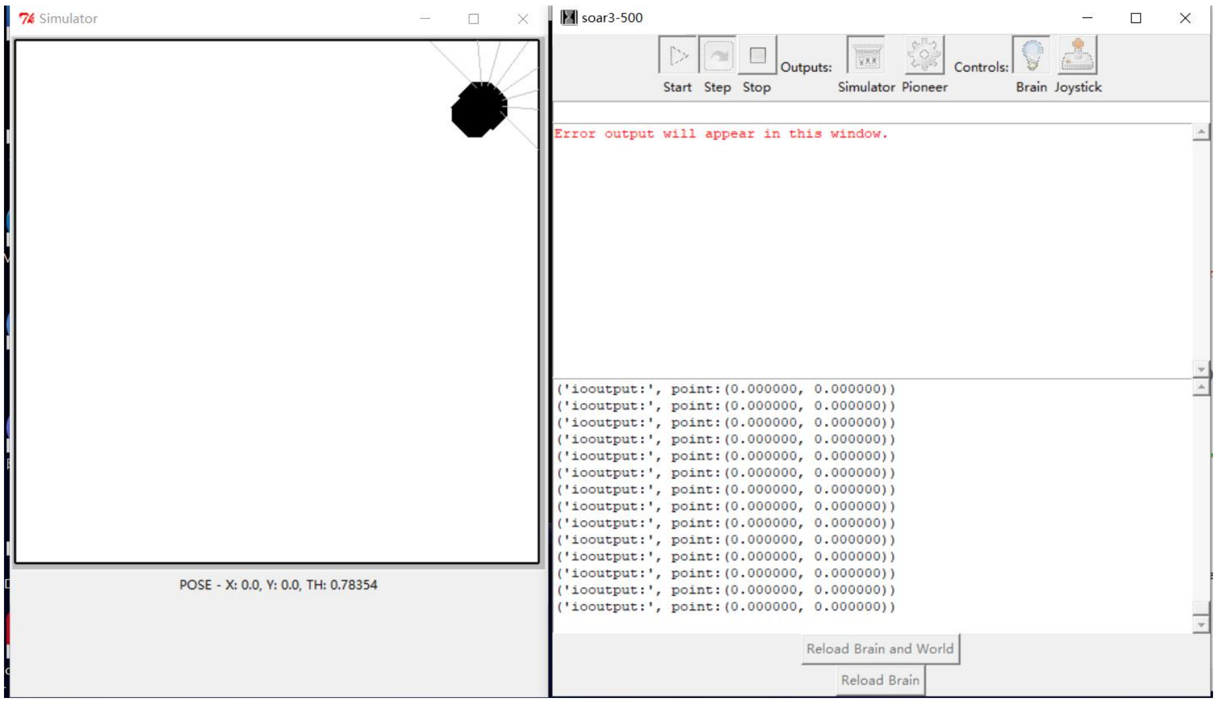


```
class stop(sm.SM):
    def __init__(self):
        self.startState = '0'
    def getNextValues(self, state, inp):
        if 1:
            return ('0',io.Action())

def statemachine():
    return sm.Cascade(ffSkeleton.FollowFigure(squarePoints),dynamicMoveToPointSkeleton.DynamicMoveToPoint())

mySM=sm.Switch(lambda inp:min(inp.sonars)<0.3,stop(),statemachine())
```

Figure 13 code

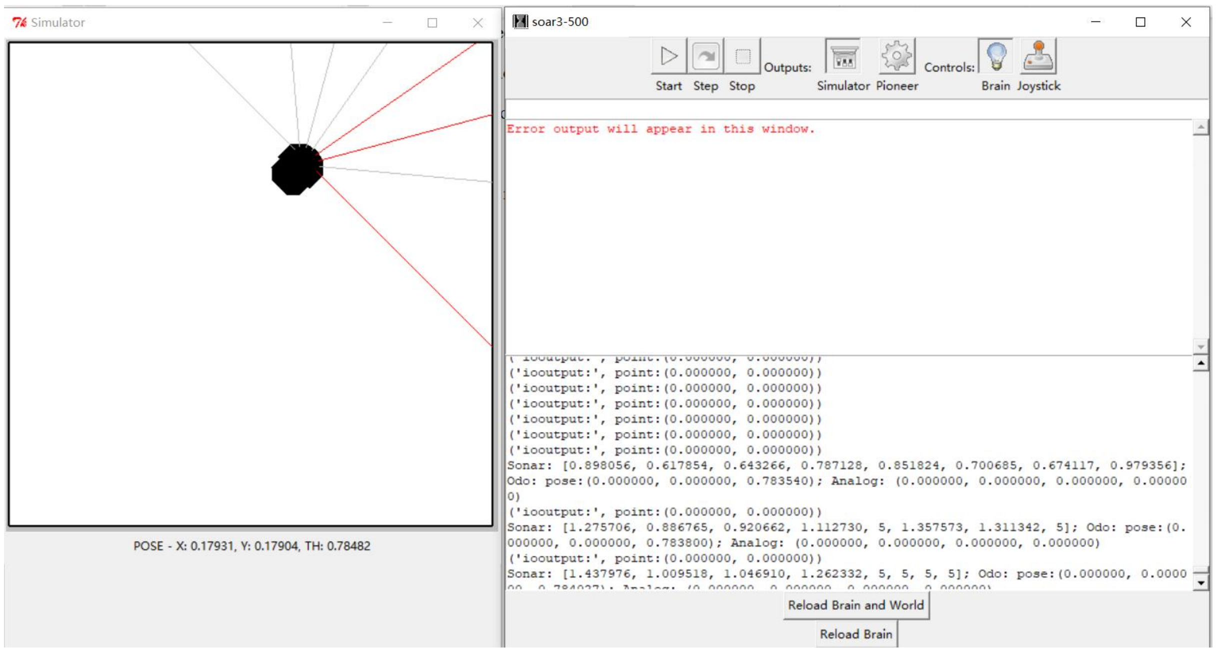
Checkoff 3. Demonstrate your safe figure-follower to a staff member. Mail your code to your partner.

Effects are as follows:



```
Error output will appear in this window.
```

```
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
```



```
Error output will appear in this window.
```

```
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
('iooutput:', point:(0.000000, 0.000000))
Sonar: [0.898056, 0.617854, 0.643266, 0.787128, 0.851824, 0.700685, 0.674117, 0.979356];
Odo: pose:(0.000000, 0.000000, 0.783540); Analog: (0.000000, 0.000000, 0.000000, 0.00000
0)
('iooutput:', point:(0.000000, 0.000000))
Sonar: [1.275706, 0.886765, 0.920662, 1.112730, 5, 1.357573, 1.311342, 5]; Odo: pose:(0.
000000, 0.000000, 0.783800); Analog: (0.000000, 0.000000, 0.000000, 0.000000)
('iooutput:', point:(0.000000, 0.000000))
Sonar: [1.437976, 1.009518, 1.046910, 1.262332, 5, 5, 5, 5]; Odo: pose:(0.000000, 0.0000
```

Figure 14 result of safe figure-follower

Summary

1.At the beginning of this work, the laboratory did not understand the cascade of the state machine well, and both changed the state and moved the position in the file, which did not meet the requirements.

2.In step 6, the use of the sm.Switch was not very skilled, and the function was realized by imitating the example in the handout.

3.And at the beginning of the definition of the stop state machine, put it in the ffSkeleton.py, because the file does not import the library function io, resulting in the car not responding in the simulation, Then,the stop state machine is put in the main program, and ensure that the ffSkeleton.py does not introduce a new library.

This experiment mainly deepened our understanding of the link between state machines.