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：Hitting the Wall

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 10 月 19 日

Introduction

Wk 4.2.1 Difference Equations

Solving logic For each system, determine a difference equation (with a finite number of terms).

• For the first system, the output is the sum of all inputs from the beginning up to and including time n, i.e. $y[n] = \sum_{i=0}^{n} x[i]$.

• For the second system, the output is the sum of all inputs from the beginning up to and including time n-1, i.e. $y[n] = \sum_{i=0}^{n-1} x[i]$.

• For the third system, the output is the sum of all scaled inputs (each input scaled by 0.1) from the beginning up to and including time n-1, i.e. $y[n] = 0.1\sum_{i=0}^{n-1} x[i]$.

Here is our answer

1. $y[n] = y[n-1] + x[n]$
2. $y[n] = y[n-1] + x[n-1]$
3. $y[n] = y[n-1] + 0.1x[n-1]$

Difference equations for wall finder

Wk 4.3.1 Wall Finder

Check Yourself 1

Solving logic At time step 0, the car's velocity is $v[0] = 1$, and its distance from the wall is $d_o[0] = 3$.According to the problem, the car maintains the same velocity $v[0]$ and, after one time step, reaches time step 1. At that point, the distance from the wall becomes $d_o[1] = d_o[0] - Tv[0] = 2.9$.

Here is our answer

$$d_o[1] = 2.9$$

Checkoff 1

Solving logic For the controller, according to the problem statement, the velocity at time n is given by $v[n] = ke[n]$, where $e[n]$ is the difference between the desired distance di[n] and the sensed distance $d_s[n]$. Therefore, we can derive:

$$v[n] = k(d_s[n] - d_i[n]).$$

For the model of the plant, the actual distance between the car and the wall , $d_o[n]$, is determined based on $d_o[n-1]$, subtracting the distance the car moves in one time step. Since the car maintains a constant velocity until it receives the next command, we subtract $Tv[n-1]$. Therefore, we can derive the equation:

$$d_o[n] = d_o[n-1] - Tv[n-1]$$

For the model of the sensor, the distance sensed by the car's sensor is delayed by one time step compared to the actual distance from the car to the wall. That is:

$$ds[n] = do[n-1].$$

For the combined difference equation for the system, we replace do[n] with Do and di[n] with Di, and substitute the delay operator R for the delay. Therefore, the sensed distance becomes $d_s[n] = RD_o$, and the velocity at time n is $v[n] = kD_i - kd_s[n] = kD_i - kRD_o$. Substituting these two equations into the equation for do[n], we get:

$$D_o = RD_o - kRD_i + kR^2D_o$$

Then convert back to original form:

$$d_o[n] = d_o[n-1] - kd_i[n-1] + kd_o[n-2]$$

Since we require that the velocity reaches 5 m/s when the target is 1 meter in front of the robot, we can calculate k=5.

Here is our answer

• **the controller** $v[n] = k(d_s[n] - d_i[n])$.

dCoeffs (input)：k , -k

cCoeffs (output):1

• **the model of the plant** $d_o[n] = d_o[n-1] - Tv[n-1]$

dCoeffs (input)：0 , -0.1

cCoeffs (output):1, -1

• **the model of the sensor** $d_s[n] = d_o[n-1]$

dCoeffs (input)：0 , 1

cCoeffs (output):1

- **The combined difference equation for the system**

$$d_o[n] - d_o[n-1] - kTd_o[n-2] = -kTd_i[n-1]$$

dCoeffs (input)：0 , -0.5 , 0

cCoeffs (output):1 , -1 , -0.5

State machines primitives and combinators

Wk.4.3.3 Do tutor problem Wk.4.3.3 (State machine composition)

System construct logic:

When the feedback path is a delay and the forward (output) path is a gain, the current output $y[n]$ is dependent on the current input $x[n]$.

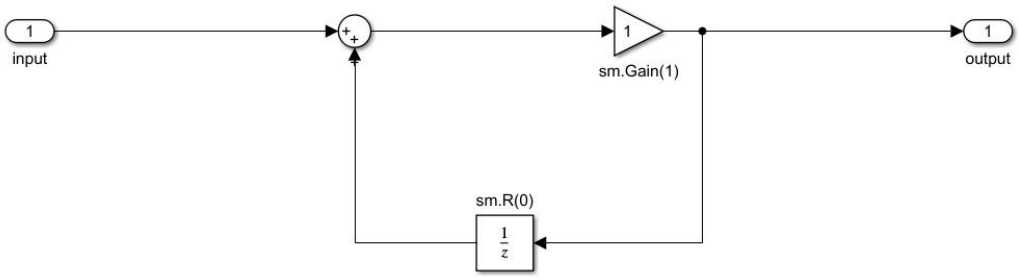

Figure 1 the system diagram for check Yourself 3

When the feedback path is a gain and the forward (output) path is a delay, the current output $y[n]$ is dependent on the previous input $x[n-$

1].

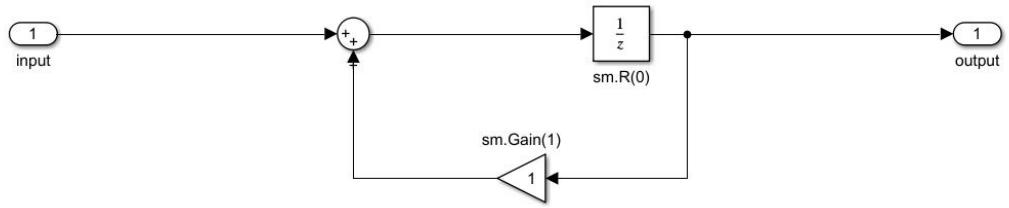

Figure 2 the system diagram for check Yourself 4

To scale the input signal, a gain machine can be cascaded in series before the input of this system

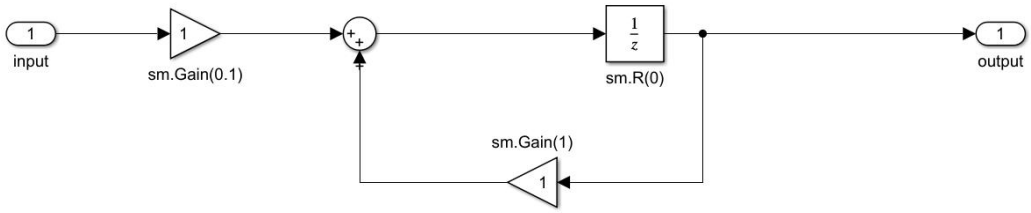

Figure 3 the system diagram for check Yourself 5

Checkoff2: Finish the system diagrams of Controller, Plant and Sensor

For Controller

$$v[n] = ke[n]$$

The velocity output is obtained by applying a gain of k to the input. Therefore, the system is constructed using sm.Gain(k)

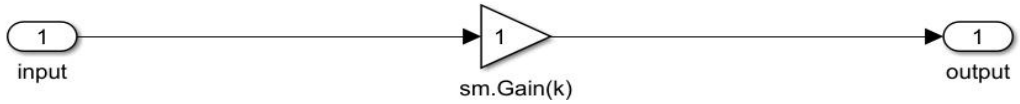

Figure 4 the system diagram for the Controller

For Plant

$$\boldsymbol{d_o[n] = d_o[n-1] - Tv[n-1]}$$

The output distance is obtained by multiplying the input velocity by the step time -T, and then feeding this result into the accumulator. **But** notice that there is one symbol change in the following step, the system is implemented by cascading sm.Gain(T) with the summation module.

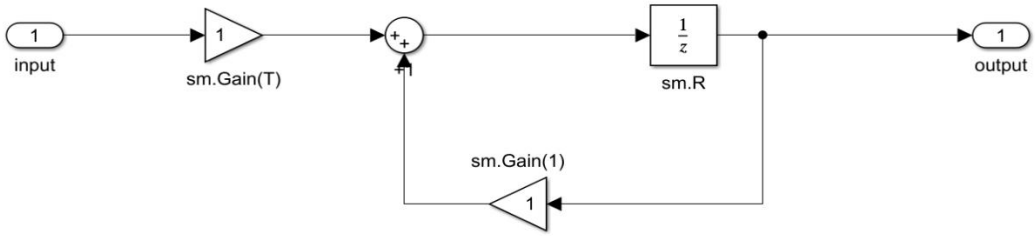

Figure 5 the system diagram for the Plant

For Sensor

$$\boldsymbol{d_s[n] = d_o[n-1]}$$

This is a simple delay unit. The output is the input from the previous time step.

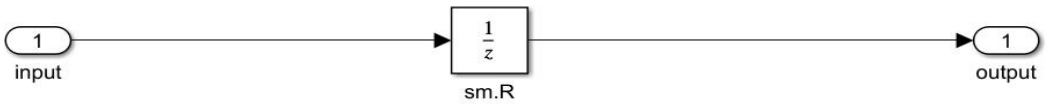

Figure 6 the system diagram for the Sensor

Constructing all

Given that the input is di[n] and the target error expression is $e[n] = d_o[n-1] - d_i[n]$, we made a small modification.

First, we changed the negative feedback path to feed back $d_o[n-1]$. Subsequently, we inverted the gain in the Plant section from -T to T.

This change effectively creates a "double negative," which precisely offsets our first modification. With these adjustments, the construction of the entire distance-keeping system is complete

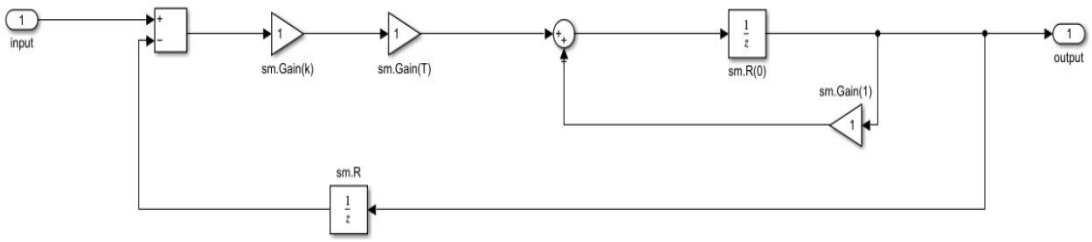

Figure 7 the complete system diagram for the Wall Finder

Check Yourself 10

We have created a discrete-time model of a robot distance control system, with the aim of moving the robot from an initial distance of 1.5 meters to and maintaining the desired distance of 0.7 meters. And by controlling three different k values, one of them makes the distance monotonically converge, one makes the distance oscillate and converge, and the other makes the distance oscillate and diverge.

Here is the key code

```
1. def plant(T, initD):

2.     Feedback_add = sm.FeedbackAdd(sm.R(initD),sm.Gain(1))

3.     return sm.Cascade(sm.Gain(T),Feedback_add)

4.
```

```
5. def controller(k):
6.     return sm.Gain(k)
7.
8. def sensor(initD):
9.     return sm.R(initD)
10.
11. def wallFinderSystem(T, initD, k):
12.     Forward_sm = sm.Cascade(controller(k),plant(T,initD))
13.     Feedback_sub = sensor(initD)
14.     return sm.FeedbackSubtract(Forward_sm,Feedback_sub)
15.
```

Through practical calculation, when the value of k is 2, the pole is on the positive real axis, there is no oscillation, and the distance converges steadily and slowly from 1.5 meters to 0.7 meters, satisfying the condition of monotonic convergence.

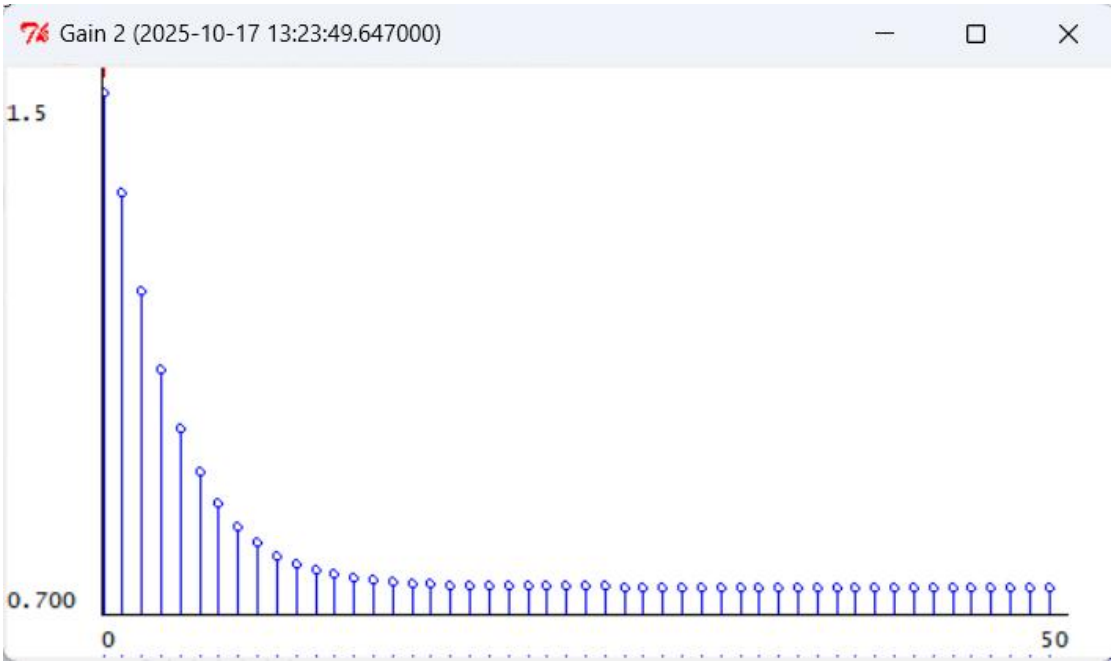

Figure 8 The situation when k=2

When k takes the value of 7, the pole is on the negative real axis, oscillating but converging, swinging back and forth on both sides of the target distance of 0.7 meters, satisfying the condition of oscillation convergence.

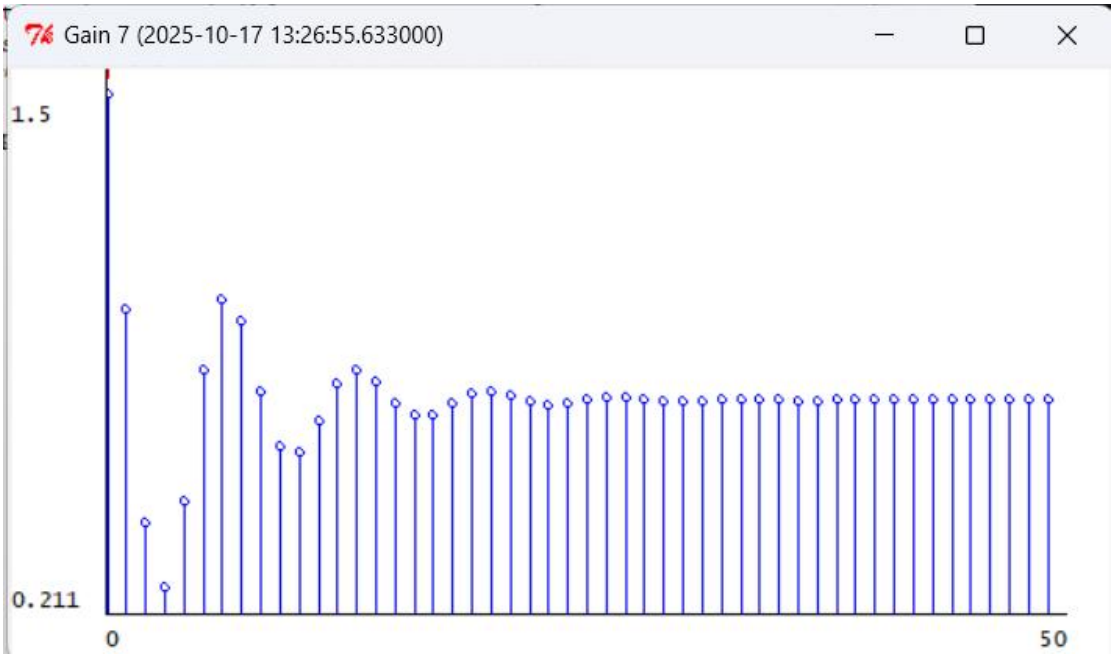

Figure 9 The situation when k=7

When the value of k is 12, the pole lies on the negative real axis outside the unit circle, oscillating and diverging. The amplitude continuously increases with time, satisfying the condition of oscillation and divergence.

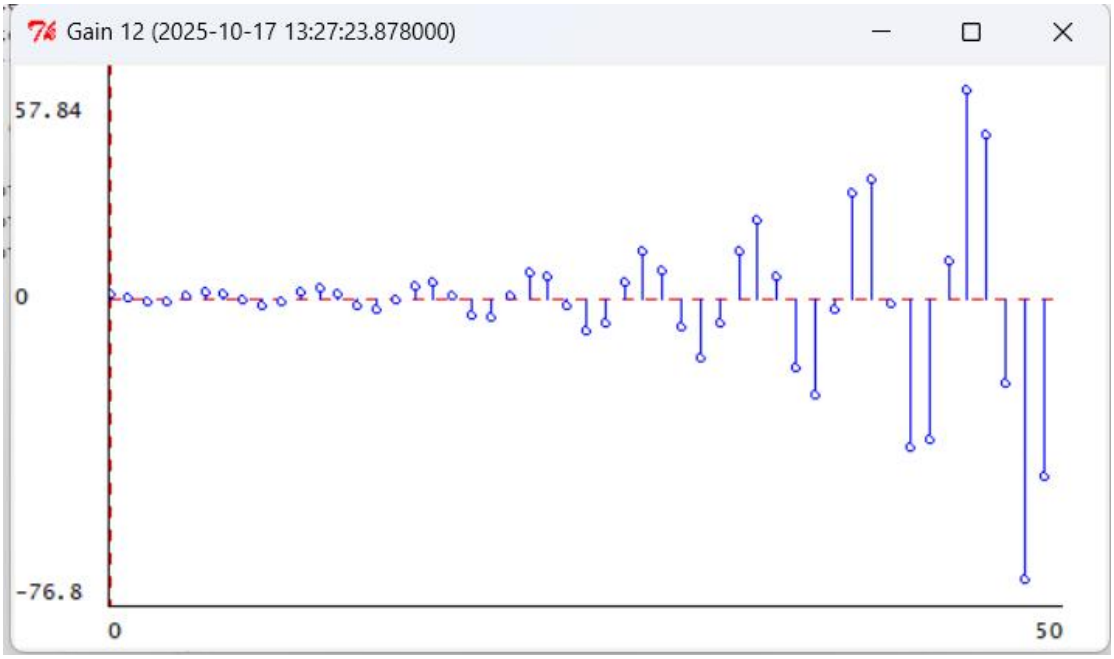

Figure 10 The situation when k=12

On the stimulated robot

Make a state machine to get sensor's input

Code logic:

The DESIRED_DISTANCE constant is defined with a value of 0.7. It sets the ideal distance between the robot and the wall

The DistanceSensor state machine class is implemented to process distance sensor data. During initialization, it constructs the initial state using a specified initial distance and number of delays to ensure valid output.

In the getNextValues method, it retrieves and prints the third sonar data from the input, updates the state by adding the latest distance data to the front of the state list and removing the oldest data from the end, and finally outputs the last element of the state list, achieving rolling caching and output of sensor data.

```
1. DESIRED_DISTANCE = 0.7 # set ideal distance to the wall
2.     class DistanceSensor(sm.SM):
3.         def __init__(self, initial_distance, num_delays):
4.             self.startState = [initial_distance] * num_delays #set the initial state to
make sure that the output is effective
5.
6.         def getNextValues(self, state, _input):
7.             print("DATA FROM SONAR3:", _input.sonars[3])  #print the third sonar's
statistics
8.             return ([_input.sonars[3]] + state[:-1], state[-1])#add the newest
distance and delete the last one
9.
```

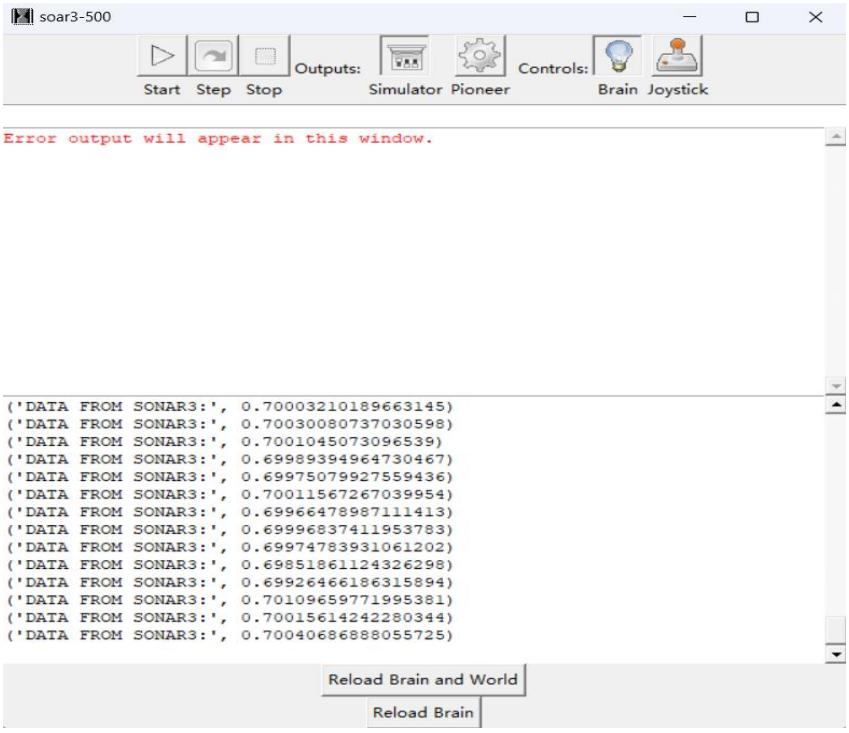

Figure 11 Printed message on the soar terminal

Make DistanceController state machine to control the robot's motion

Code logic:

The DistanceController class receives gain and maximum speed parameters during initialization.

Its getNextValues method calculates and returns the robot's action commands based on the deviation between the distance input from the sensor and the desired distance, where the rotational speed is set to 0 and the forward speed is related to the distance deviation.

```
1. class DistanceController(sm.SM):
2.     def __init__(self, gain,max_vel):
3.         self.gain = gain #set some variables(gain and max_vel)may be should do some
changes
4.         self.max_vel = max_vel#restrict robots 's max speed
5.
6.     def getNextValues(self, state, sensor_input):
7.         return (state, io.Action(rvel=0, fvel=2*(-DESIRED_DISTANCE + sensor_input)))
8.     #return the order ,rotate speed =0 ,foward speed can change according the distance
to the wall
```

Casade those two state machines

Code logic:

sm.Cascade is used to cascade DistanceSensor and DistanceController into brain_sm, forming a complete logic chain from sensor data processing to control command generation, which serves as the core control module of the robot.

```
1.   brain_sm = sm.Cascade(DistanceSensor(1.5, 1), DistanceController(gain=0.5,max_vel =
 0.5))#ccascade the two state machines and set some fixed numbers
```

```
2.   brain_sm.name = 'brainSM'
```

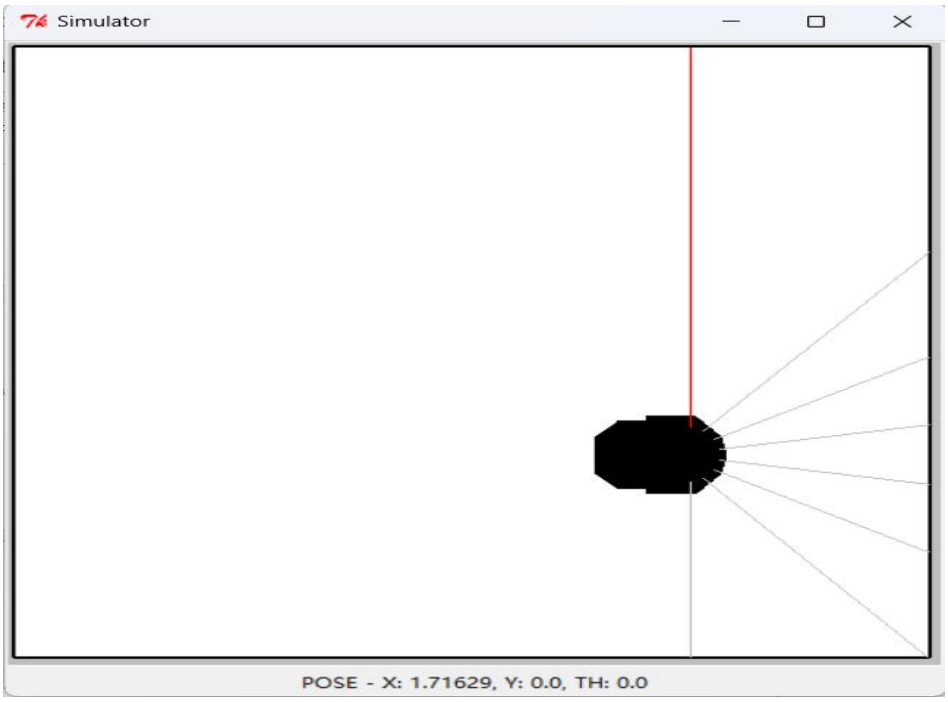

Figure 12 The Simulation of Car Operation

Check Yourself 11

Figure 13 The result in soar when k=2

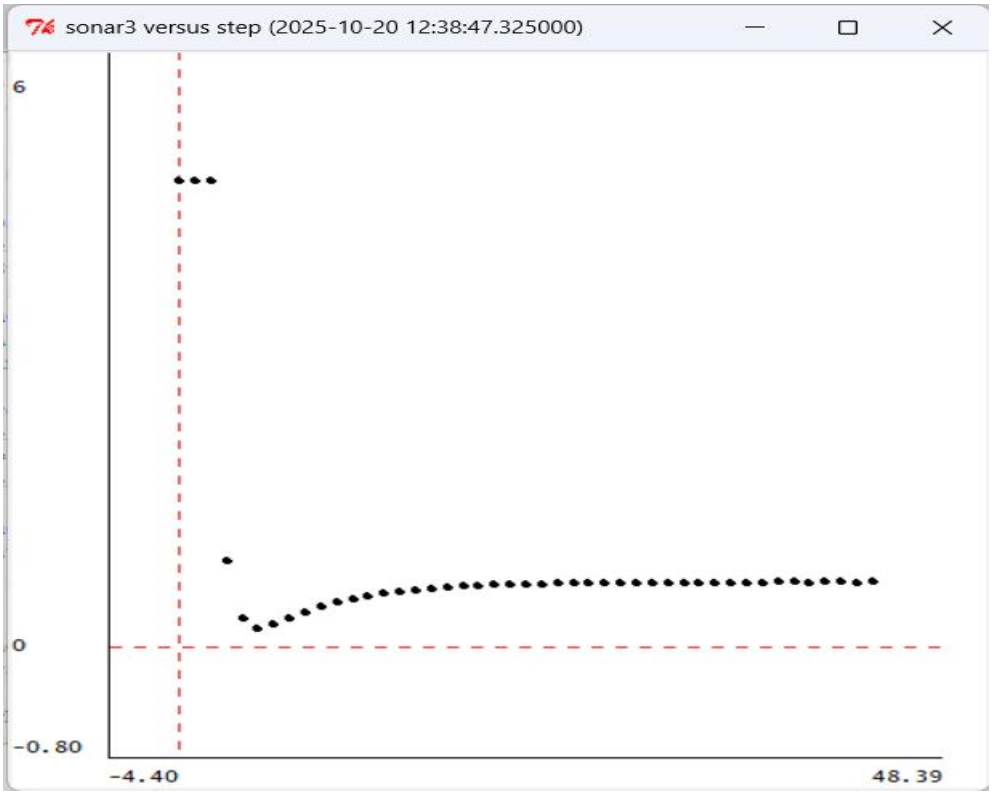

Figure 14 The result in soar when k=7

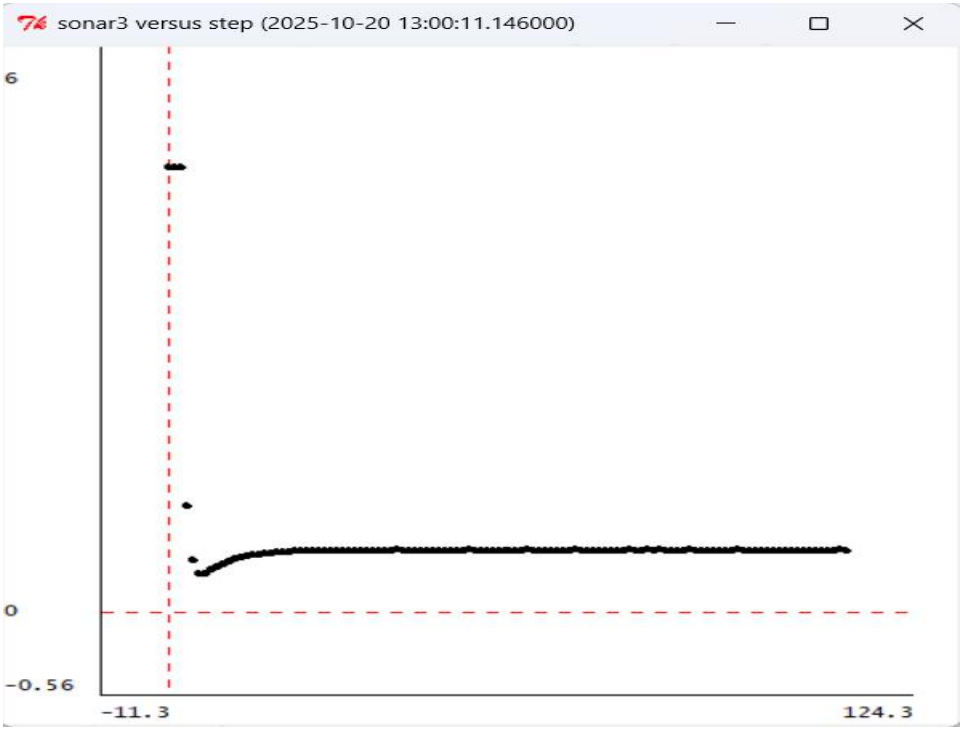

Figure 15 The result in soar when k=12

Checkoff3: Explain how they differ, and speculate about why

The theoretical model accurately predicts the system's behavior for gain values of k=2 and k=7. At k=12, however, a notable discrepancy emerges. Due to its linear nature, the theoretical model predicts divergence. In contrast, the Soar simulator incorporates a physical (non-linear) constraint, specifically the maximum velocity. This limitation effectively "clamps" the system's output, which prevents divergence and allows the system to ultimately converge despite the high gain.

Appendix: The Description of AI Usage in the Report

As it is rather difficult to read English literature, we used AI to understand the relevant concepts and issues, and also utilized the AI translation function. During the task process, we relied on AI to help us better understand the meaning of some code and state machines.