山东大学

崇新学堂

2024－2025 学年第 1 学期

实 验 报 告

课程名称:	电子信息工程导论实验
实验名称:	Design lab4　Hitting Wall

专 业 班 级	2023 崇新学堂
组　　　别	第八组 赵耀 魏天行 杨瑞 王嵩泽
实 验 时 间	2024.10.26

Goals:

In this lab, we develop a signals and systems model of that system, and use the model to understand the performance of the brain/robot system.With this model, we will be able to answer key performance questions.

This will be studied in three steps:

• Develop a difference equation model of the wall finder system

• Build a state machine model of the feedback system

• Implement a robot brain realizing the state machine controller

Step1Difference equations for wall finder

Objective: Develop a difference equation model for the wall finder system.

Make a simple model of the brain/robot system, as follows. Let do[n] (the 'o' stands for output) represent the current distance from the robot to the wall, and let d_i[n] (the 'i' stands for input) represent the (current) desired distance to the wall. Also let v represent the forward velocity of the robot. Let T = 0.1 seconds represent the time between steps.

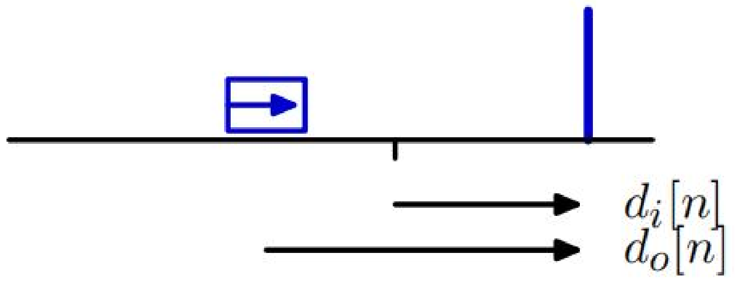


Figure1 a simple model of the system

When the robot receives a new command, we assume that the robot immediately changes its velocity and then holds the new velocity constant until it recieves the next command (i.e., the robot accelerates so fast that we can ignore the acceleration time).

Check Yourself 1&Wk.4.2.1:

Given the following conditions, what is the distance to the wall on step 1?

$v[0] = 1$ $d_o[0] = 3$

$v[1] = 2$ $d_o[1] =$2.9

Assume that the sensor measures the

current distance do[n] and generates the sensed distance ds[n], which is equal to the current distance delayed by one step time. Let e[n] represent the error signal, which is the difference between the input distance di[n] and the sensed distance ds[n]. On each step, the controller commands a forward velocity v[n] in proportion to the error so that v[n] = ke[n]. Choose k so that the velocity is 5 m/s when the desired location is 1 m in front of the robot (think about the previous figure showing the position of the robot in order to help frame this calculation for k).

Check Yourself 2&Wk.4.3.1:

Fully label the system diagram. Include d_o, d_i, d_s, v, e.

Control the robot to move to desired distance from a wall.

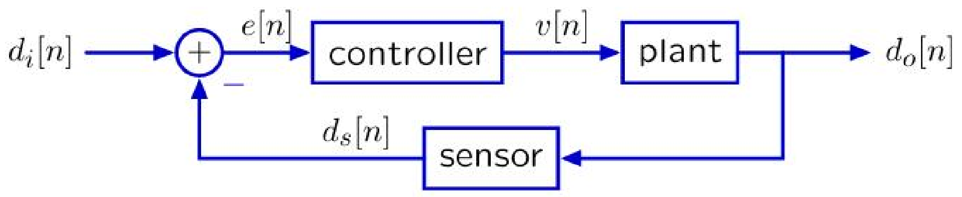


Figure2 the system diagram

Controller (brain) – sets velocity $\propto$ error:

$$v[n] = ke[n] = k\big(d_i[n] - d_s[n]\big)$$

Plant (robot locomotion) – given $v[n]$, drives to new position:

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

Sensor (sonars) – reports output, but also introduces delay:

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

Checkoff 1.&Wk 4.3.2:

Combine these equations to derive a difference equation that relates do to di, by:

1. Converting the difference equations to operator equations in R,

2. Solving for Do in terms of Di, and

3. Converting the result back to a difference equation.

• the difference equation:

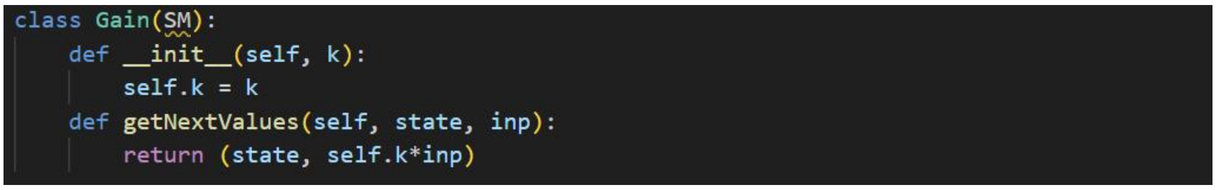
$$d_o[n] = d_o[n-1] - Tv[n\text{-}1] = d_o[n-1] - T(kd_i[n\text{-}1] - kd_s[n\text{-}1]) = d_o[n-1] - Tkd_i[n\text{-}1] + Tkd_s[n\text{-}1]$$
$$= d_o[n-1] - Tkd_i[n\text{-}1] + Tkd_o[n-1] = (1+Tk)d_o[n-1] - Tkd_i[n\text{-}1]$$
$$\Leftrightarrow d_o[n] = (1+Tk)d_o[n-1] - Tkd_i[n\text{-}1]$$

Step2 State machines primitives and combinators

Objective: Build a state machine model of the wall finder system, based on its system diagram representation.

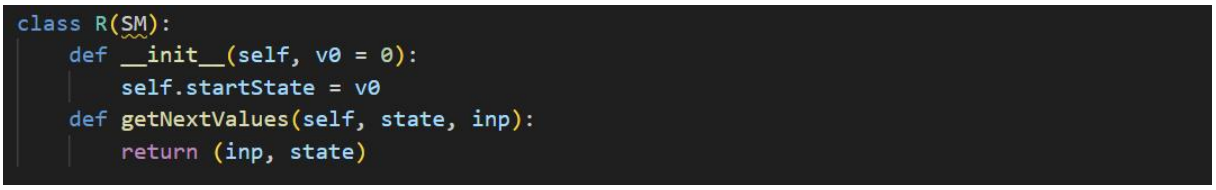
The wall finder system, as represented by the system diagram of Figure 1, can be modeled as a combination of primitive state machines, using two basic constructs: sm.Gain and sm.R. This is true because the wall finder system is a "linear time-invariant" (LTI) system.

The sm.Gain state machine is really just a pure function: the output at step n is the input at step n, times a constant, k. The state is irrelevant. The reason we create this as a type of state machine is that we want to use the principles of PCAP to be able to combine it with other state machines to create new kinds of state machines.



```
class Gain(SM):
    def __init__(self, k):
        self.k = k
    def getNextValues(self, state, inp):
        return (state, self.k*inp)
```

The sm.R state machine is a renamed version of the Delay state machine. It takes a value at initialization time which specifies the initial output of the machine; thereafter, the output at step n is the input at step n − 1.



```
class R(SM):
    def __init__(self, v0 = 0):
        self.startState = v0
    def getNextValues(self, state, inp):
        return (inp, state)
```

For the purposes of building LTI systems, the feedback addition composition and Feedback subtraction composition will be useful.If we want to apply one of the feedback operators in a situation where there is only one machine, we can use sm.Gain(1.0) or sm.Wire() as the other argument.

Check Yourself 3

Use gains, delays, and adders to draw a system diagram for the first system in tutor problem Wk.4.2.1. (That is, the tutor problem that you did before coming to lab).

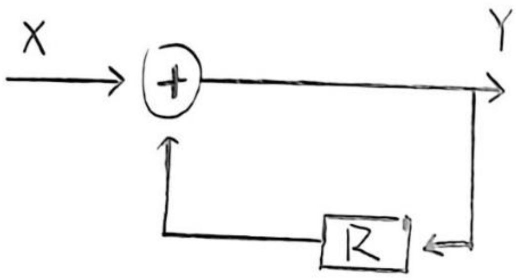


Figure3 the first system diagram

Check Yourself 4

Use gains, delays, and adders to draw a system diagram for the second system in tutor problem Wk.4.2.1.

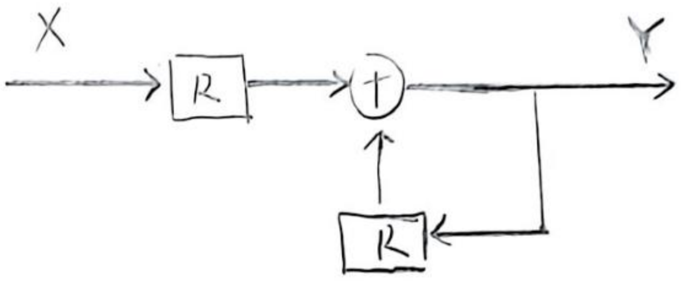


Figure4 the second system diagram

Check Yourself 5

Use gains, delays, and adders to draw a system diagram for the third system in tutor problem Wk.4.2.1.

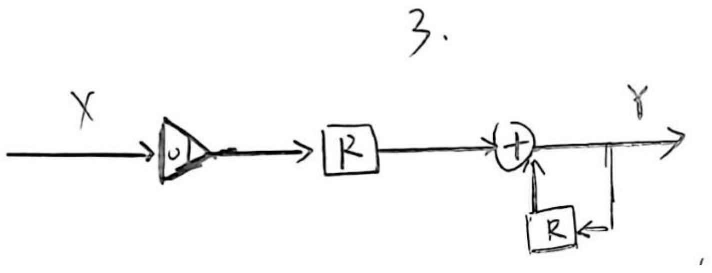


Figure5 the third system diagram

Check Yourself 6-8

Use gains, delays, and adders to draw a system diagram for the **controller, plant and sensor** in the wall-finder system.

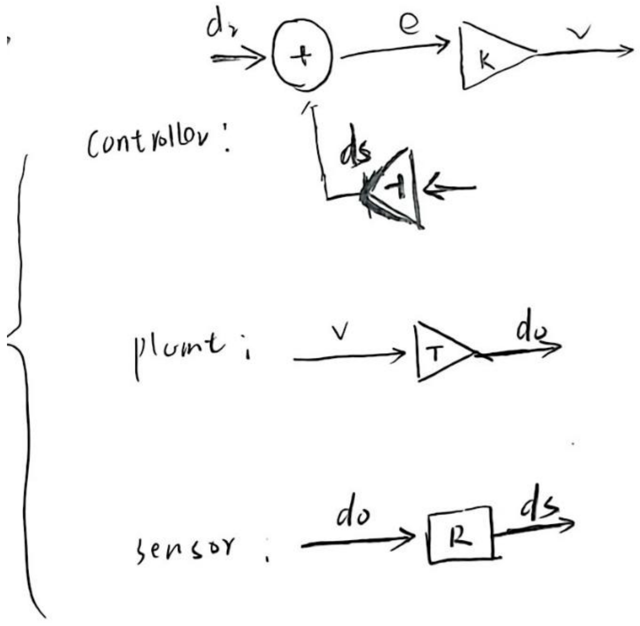


Figure6 the controller, plant and sensor system diagram

Check Yourself 9

Connect the previous three component systems to make a diagram of the wall-finder system. Label all the wires. Draw boxes around the controller, plant, and sensor components.

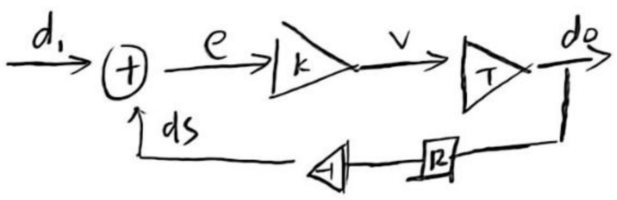


Figure7 the wall-finder system diagram

Checkoff 2

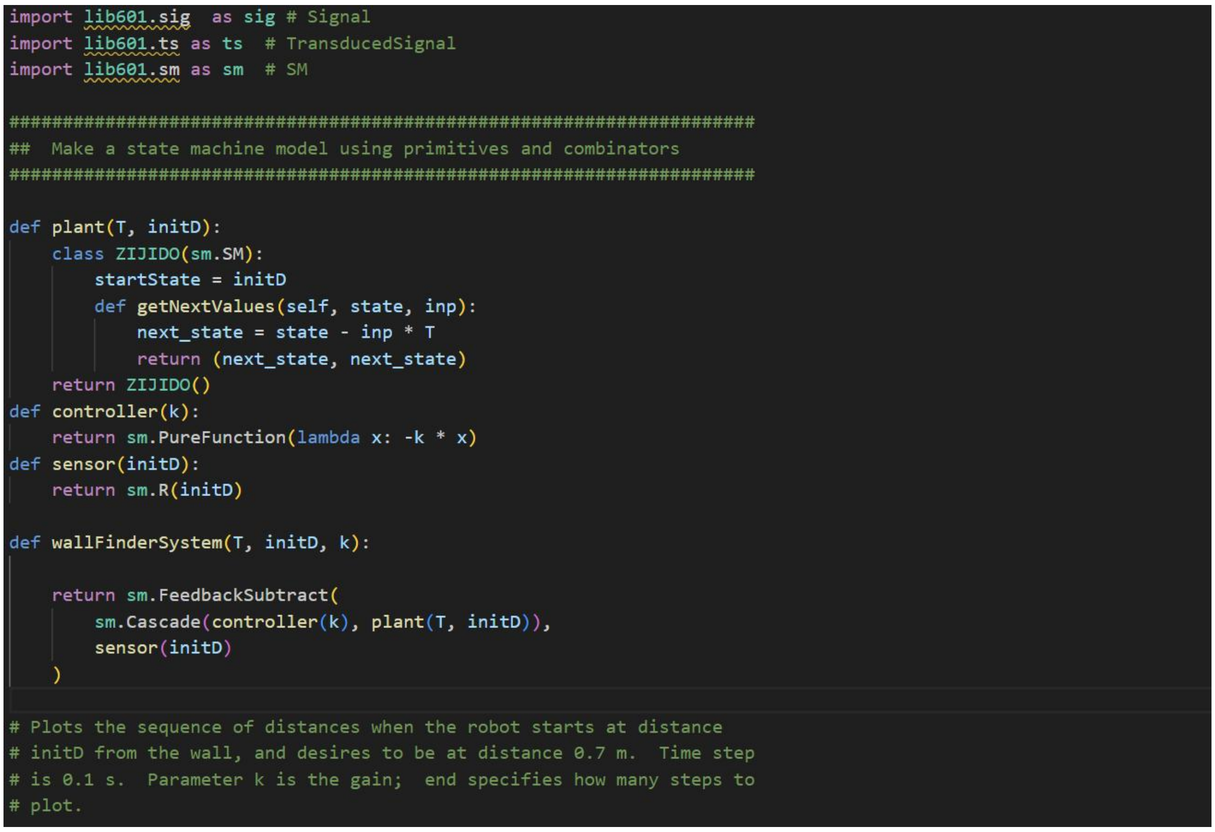
Wk.4.3.4 Explain your system diagrams to a staff member. Identify instances of cascade and feedback composition for the wall-finder system.

The diagrams of this series are presented above.

Step3

Wk.4.3.5:

Do tutor problem Wk.4.3.5.Write your code in designLab04Work.py. Each function should return an instance of state machine of the corresponding block. Test your code with > idle -n Then submit your code in the Tutor.



```
import lib601.sig  as sig # Signal
import lib601.ts as ts  # TransducedSignal
import lib601.sm as sm  # SM

######################################################################
##  Make a state machine model using primitives and combinators
######################################################################

def plant(T, initD):
    class ZIJIDO(sm.SM):
        startState = initD
        def getNextValues(self, state, inp):
            next_state = state - inp * T
            return (next_state, next_state)
    return ZIJIDO()
def controller(k):
    return sm.PureFunction(lambda x: -k * x)
def sensor(initD):
    return sm.R(initD)

def wallFinderSystem(T, initD, k):

    return sm.FeedbackSubtract(
        sm.Cascade(controller(k), plant(T, initD)),
        sensor(initD)
    )

# Plots the sequence of distances when the robot starts at distance
# initD from the wall, and desires to be at distance 0.7 m.  Time step
# is 0.1 s.  Parameter k is the gain;  end specifies how many steps to
# plot.
```

With T=0.1 and an initial distance to the wall of 1.5 meters, experiment with different values of the gain. You can do this using the plotD procedure, defined in designLab04Work.py. Use idle -n. For a given gain value, k, it will make a plot of the sequence of distances to the wall.

Check Yourself 10

Find three different values of k, one for which the distance converges monotonically, one for which it oscillates and converges, and one for which it oscillates and diverges. Make plots for each of these k values. Save screen shots (see Reference tab of the 6.01 website) for each of these plots.

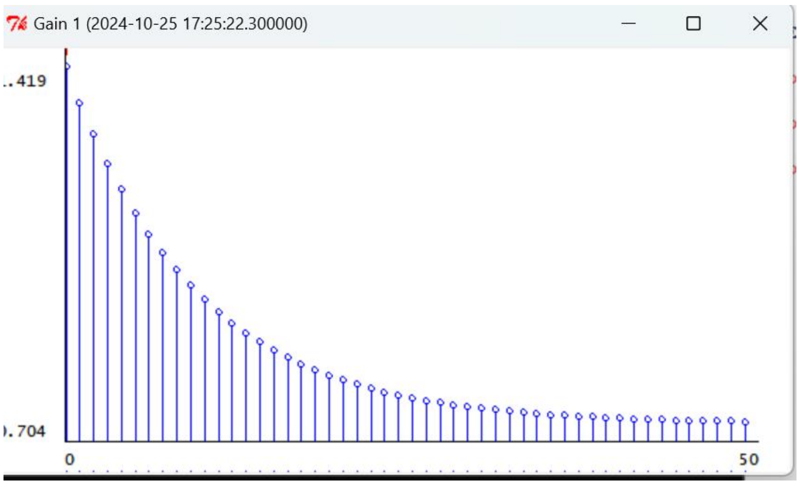


Figure8 k=1 for which the distance converges monotonically

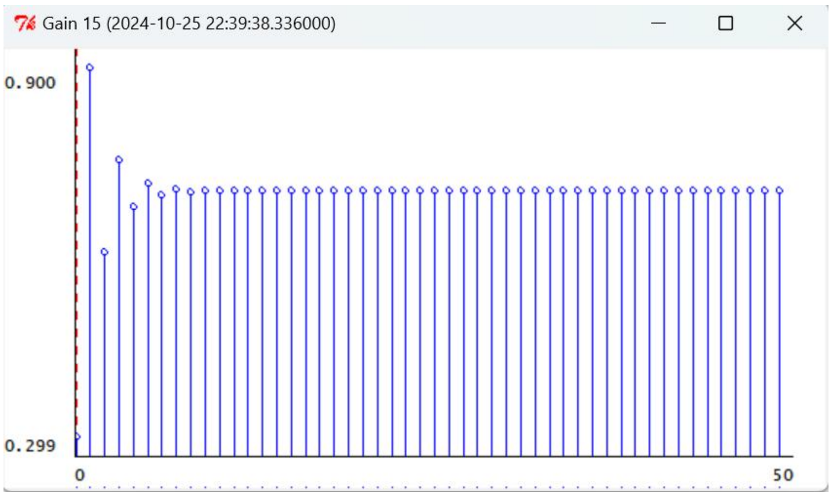


Figure9 k=15 for for which it oscillates and converges

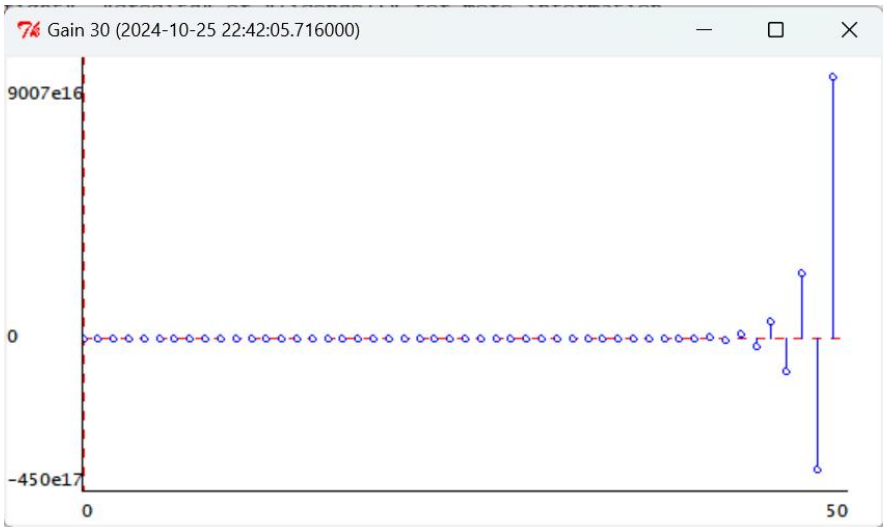


Figure10 k=30 for which it oscillates and diverges

Step3

On the simulated robot

Objective: Implement a brain for the wall-finder problem using a state machine, as described in Section 2.

Recall that the robot itself is the plant and so we do not need to write any code for that. We have already implemented a Sensor state machine which outputs a delayed version of the values of sonar sensor 3 (the robot actually has relatively little delay in its sensing).

Your job is to implement the Controller state machine, which takes as input the output of the sensor state machine, and generates as output instances of io.Action with 0 rotational velocity and an appropriate forward velocity. It should depend on dDesired, which is the desired distance to the wall. Do this by editing the getNextValues method of the Controller class in Desktop/6.01/designLab04/smBrainPlotDistSkeleton.py. Your controller should attempt to make the output of sonar sensor 3 be equal to 0.7, even though sensor 3 doesn't point straight forward.

Check Yourself 11

For each of the three gains you found Check Yourself 10, run the simulated robot in the wallFinderWorld.py world, and save the plots.

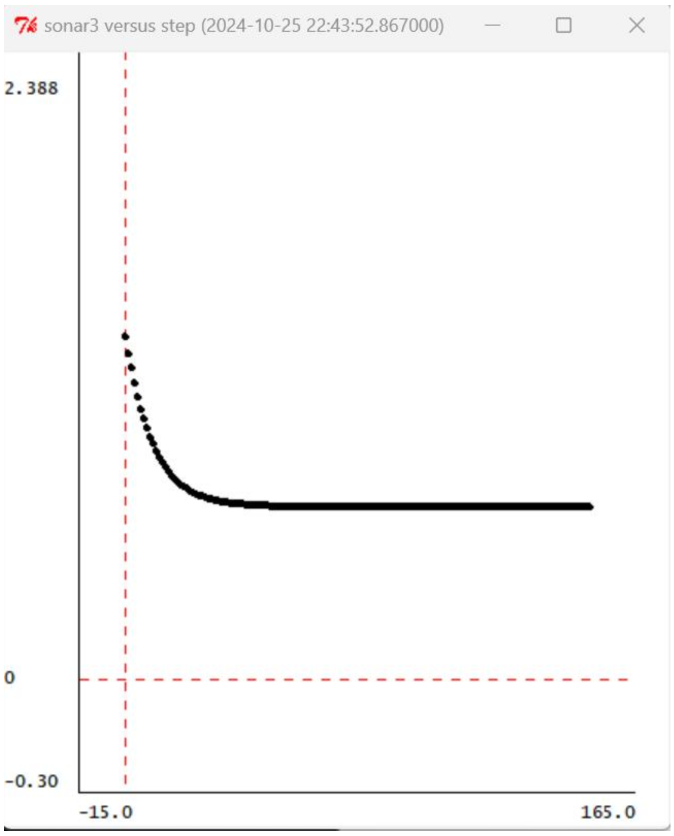


Figure11 k=1

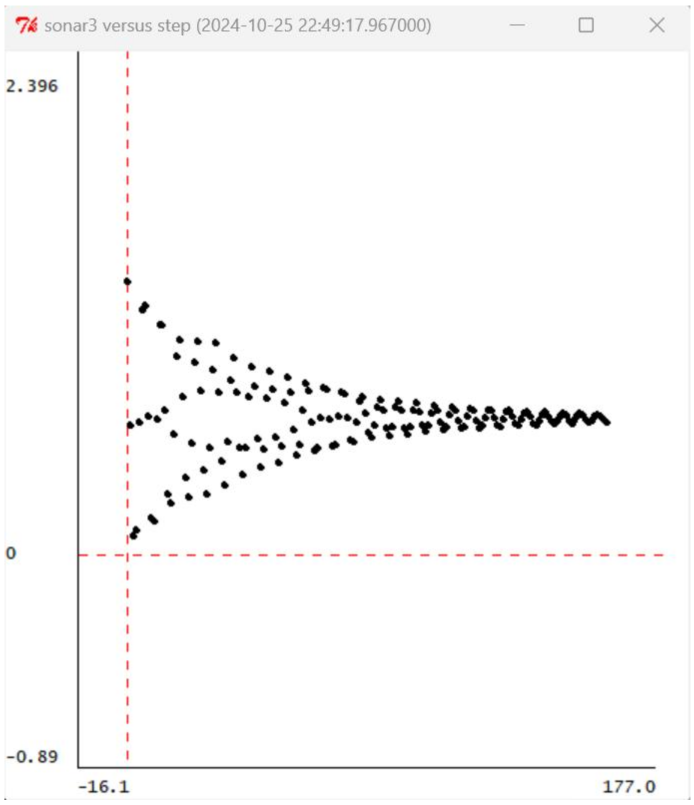


Figure12 k=15

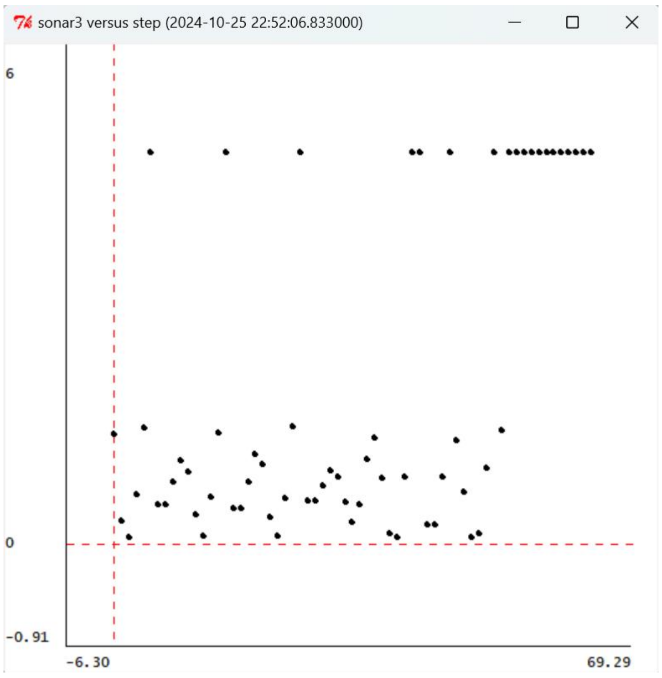


Figure13 k=30

Checkoff 3

Wk.4.3.7

Compare the plots from Check Yourself 10 and 11. Explain how they differ, and speculate about why. Email your code and plots to your partner.

Possible cause

1. The sonar system that is positioned at the front of the robot does not perfectly align with the exact centerline directly in front of the robot, resulting in discrepancies between the actual distance from the robot to the wall and the distance measured by the sonar system. These discrepancies arise because the sonar system's readings may be slightly off to one side or the other, causing inaccuracies in the data it provides.

2. In the model used for Checkoff10, the acceleration phase of the robot's movement and the duration of data processing are not taken into account, which could potentially result in alterations to the values of time (T) and velocity (V). By neglecting these factors, the model might not accurately reflect the true dynamics of the robot's motion and the precise timing of data handling, leading to possible miscalculations in these critical variables.

Summary

After several design-labs, our programming skills have improved, making it easier to complete this experiment. During the experiment, however, we noticed that the differential equation of the car system we derived did not precisely match that simulated in Soar. This discrepancy suggests that certain conditions of the model were not taken into account. If these factors were to be amplified during actual vehicle operation, it would be imperative to consider these error factors.