Check Yourself 1. **What should be the types of the input to and the output from a state machine of the WallFollower class? What should be the sign of k?**

The input is the actual distance to the right wall and the output is the next action of the robot including its forward velocity and rotational velocity. The sign of k is Gain.

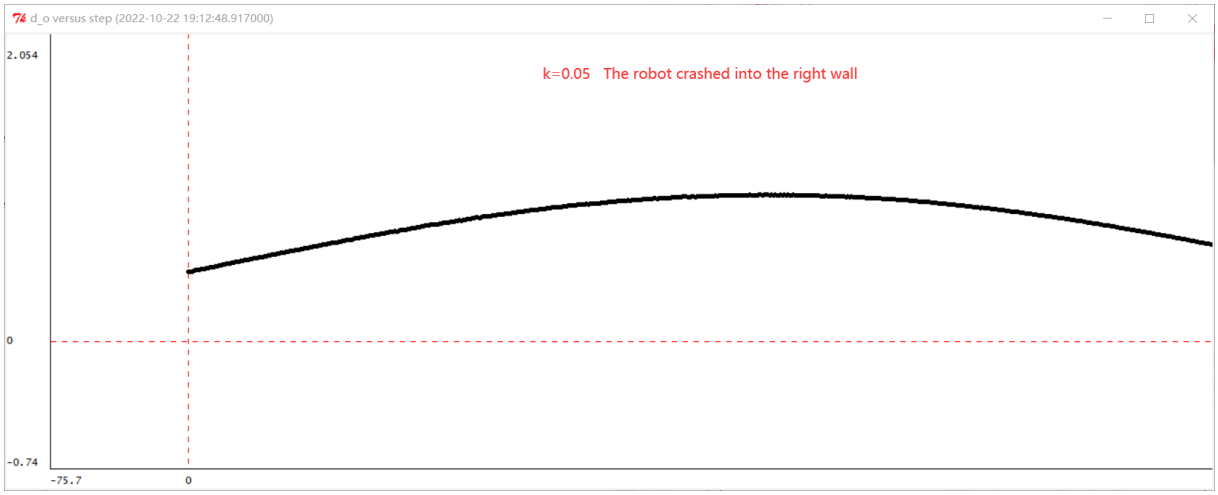
Step1

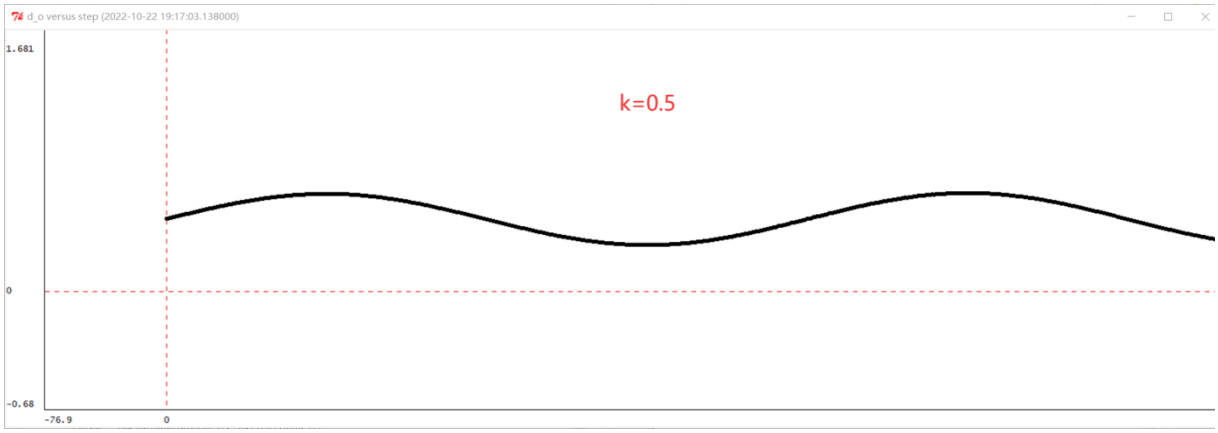
My code for constructing this composite machine into propWallFollowBrainSkeleton.py

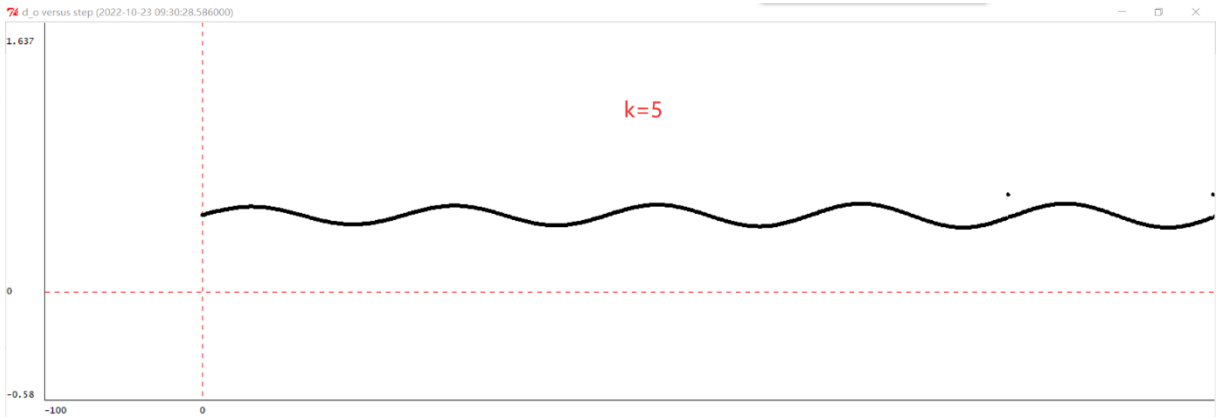
```
import math
import lib601.sm as sm
from soar.io import io
import lib601.gfx as gfx
import lib601.util as util
import lib601.sonarDist as sonarDist

import lib601.ts as ts
import lib601.sig  as sig


import operator

reload(gfx)

##############################################################################
#
#            Brain SM
#
##############################################################################

desiredRight = 0.5
forwardVelocity = 0.1
Gain=50

# No additional delay
class Sensor(sm.SM):
    def getNextValues(self, state, inp):
        return (state, sonarDist.getDistanceRight(inp.sonars))

# inp is the distance to the right
class WallFollower(sm.SM):
    def getNextValues(self, state, inp):
        k=Gain
        print(inp)
        return (state,io.Action(fvel = forwardVelocity,rvel = k*(desiredRight -

mySM = sm.Cascade(Sensor(), WallFollower())

##############################################################################
#
#            Running the robot
#
##############################################################################

def plotDist():
    func = lambda: sonarDist.getDistanceRight(io.SensorInput().sonars)
    robot.gfx.addStaticPlotFunction(y=('d_o', func))

def setup():
    robot.gfx = gfx.RobotGraphics(drawSlimeTrail=False)
    plotDist()
    robot.behavior = mySM
    robot.behavior.start(traceTasks = robot.gfx.tasks())

def step():
    robot.behavior.step(io.SensorInput()).execute()

def brainStart():
    # Do this to be sure that the plots are cleared whenever you restart
    robot.gfx.clearPlotData()

def brainStop():
    pass

def shutdown():
    pass
```

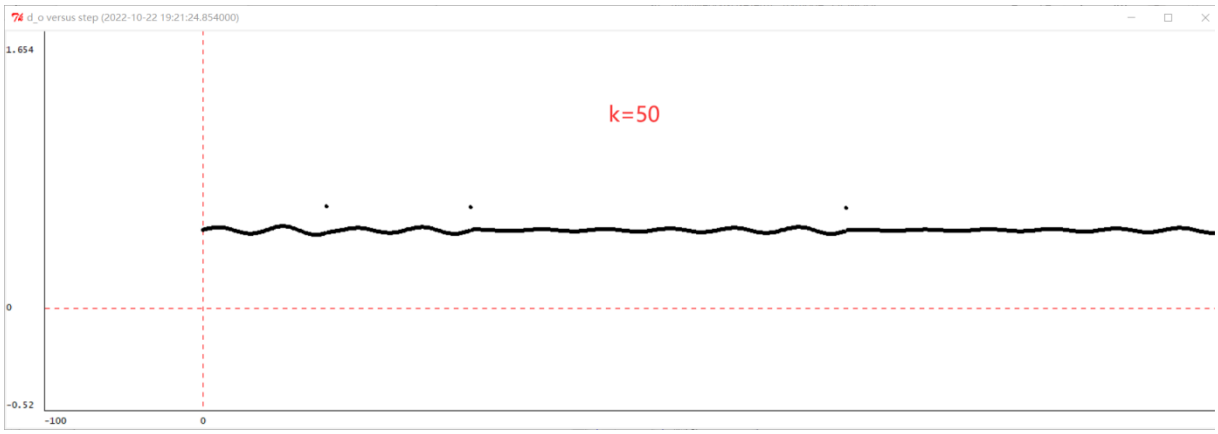
Step2

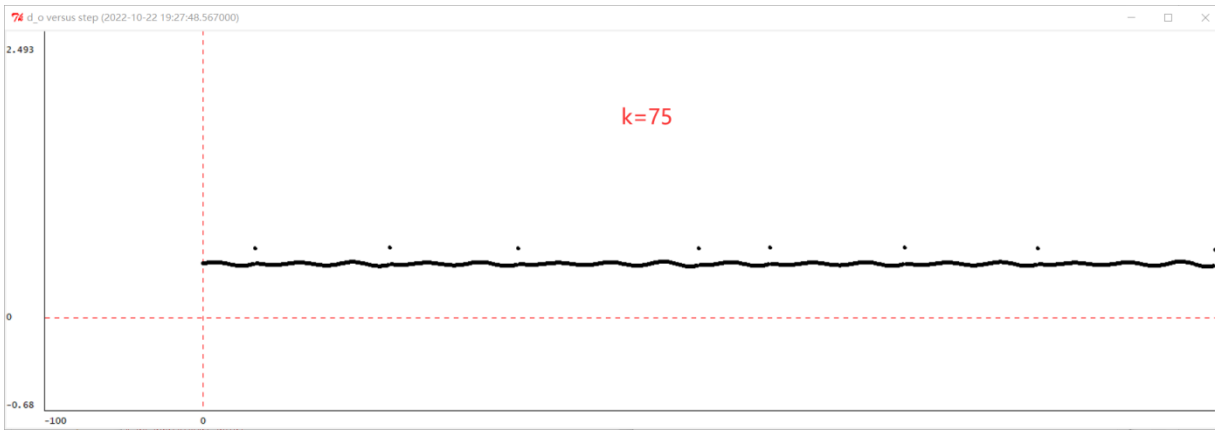
Change k to observe the movement of the robot.

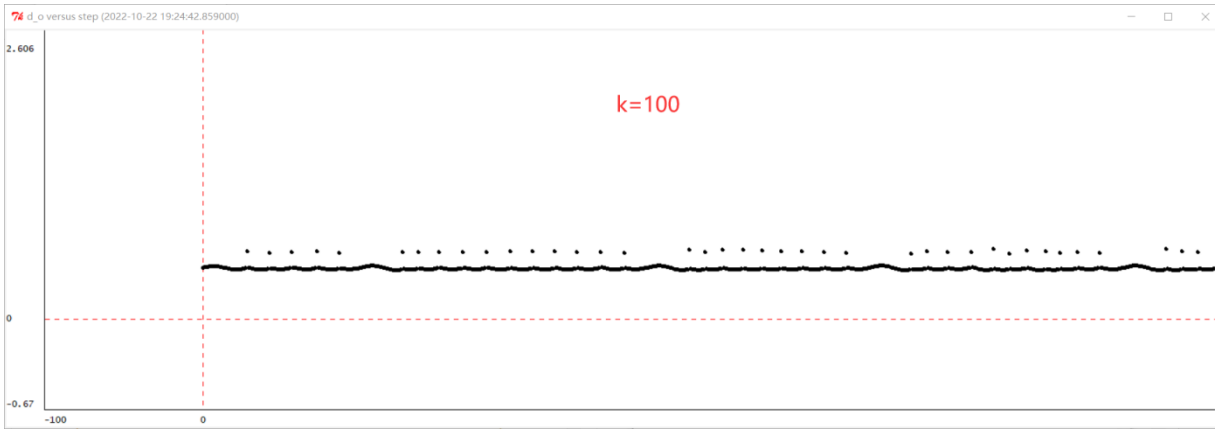


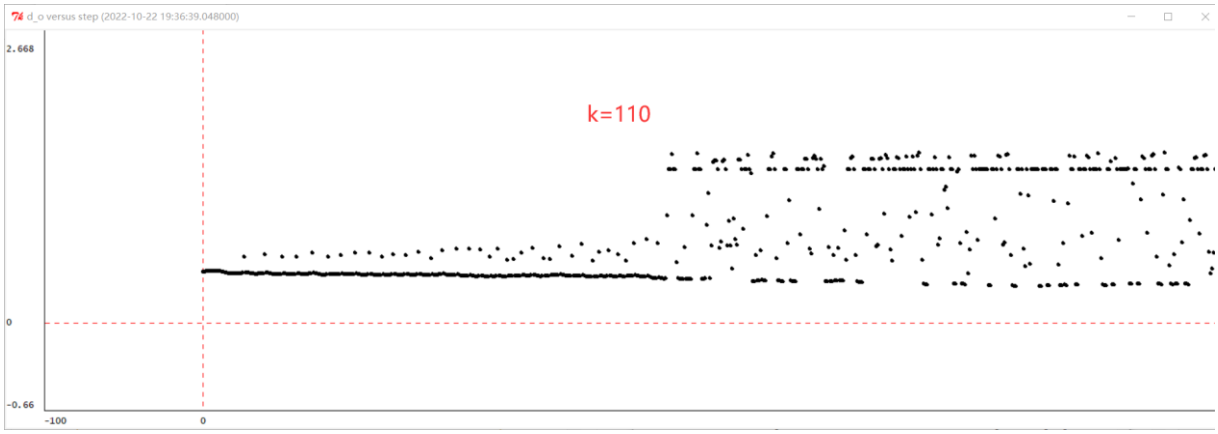





d_o versus step (2022-10-22 19:21:24.854000)
1.654
k=50
0
-0.52
-100
0


d_o versus step (2022-10-22 19:27:48.567000)
2.493
k=75
0
-0.68
-100
0


d_o versus step (2022-10-22 19:24:42.859000)
2.606
k=100
0
-0.67
-100
0


d_o versus step (2022-10-22 19:36:39.048000)
2.668
k=110
0
-0.66
-100
0

Checkoff 1. **Show your plots to a staff member. Describe how the gain k affects behavior of the wall-follower. For what value of k (if any) does the distance to the wall converge most rapidly to the desired value of 0.5 m? Compare the effect of k in this lab and in the lab 4 (wall finder).**

k influences the behavior of the robot by influencing the rotating speed of the robot. When k is greater than 0 and less than 100, The larger k is, the smaller the oscillation amplitude of the robot near the desired value is. When k is greater than 100, with the increase of k, the robot will shake heavily and move slowly because of too fast speed.

We see that the function doesn't converge for any value of k. When k is equal to 75, because the amplitude of the behavior of the system is small, it can be approximated as convergence.

In lab 4, k controls the magnitude of the forward speed and in lab 5, k controls the magnitude of the angular speed. In lab 4, we will find a k that makes the system behavior converge quickly to the expected value when we change the value of k, but in lab 5, we cannot find such a k value that makes the function converge quickly.

Step3

Controller model: Assume that the controller can instantly set the rotational velocity $\omega[n]$ to be proportional to the error $e[n]$. Express this relation as a difference equation.

$$\omega(n)=k*e(n)$$

Step4

- **Plant 1:** Write an expression for $\theta[n]$, the robot's angular orientation with respect to the wall, that depends on its rotational velocity. Assume that the rotational velocity at time $n-1$ is $\omega[n-1]$, and that this rotational velocity is constant until time n.

$$\theta(n)-\theta(n-1)=\omega(n-1)*T$$

- **Plant 2:** Write an expression for $d_o[n]$, that depends on angular displacement. Assume that the angular displacement at time $n-1$ is $\theta[n-1]$, and that this angular displacement is constant until time n.

  Linearize the resulting difference equation using the small angle approximation (i.e., if θ is small, then $\sin\theta \approx \theta$). This approximation makes our model linear, allowing us to analyze it easily. It may be useful, however, to think about the consequences of this approximation when trying to account for behaviors in subsequent sections.

$$d_o(n)-d_o(n-1)=\theta(n-1)*VT$$

Sensor model: To keep things simple, we will model the sensor as a wire: that is, assume it introduces no delay.

The subsystems represented by the three difference equations above connect together to form a system of the following form. Label each wire in the block diagram with the name of the corresponding signal. You may also find it useful to label each of the boxes in this diagram with the element of the model (controller, plant1, plant2) corresponding to each of the three difference equations you derived.

Step5

We first transform the above block diagram into operator expressions.

Controller: $U = kX \Rightarrow \frac{U}{X} = k$

Plant1: $W - RW = TR * U \Rightarrow \frac{W}{U} = \frac{TR}{1-R}$

Plant2: $Y - RY = VT * RW \Rightarrow \frac{Y}{W} = \frac{RTV}{1-R}$

Sensor：

$$VT^2kR(X-Y)+VT^2kR^2(X-Y)=Y$$

We then transform this equation into a system function.

$$H=\frac{D_0}{D_i}=\frac{Y}{X}=\frac{VT^2kR^2}{VT^2kR^2+(1-R)^2}$$

Wk.5.3.2.

Problem Wk.5.3.2: Wall Follower System Function

Enter the system function of the proportional wall follower $H=\frac{D_o}{D_i}$ by entering the numerator and denominator polynomials in R. Enter the polynomials as expressions the way you would enter them in Python (although dropping * as in normal algebraic notation should also work). You can use the following symbols:

```
R, T, V, k
```

(these variables are case-sensitive)

An example of a polynomial in R entered in this format is:

```
k*T*(R**2) - V*(T**2)*R + 1
```

- You need to enter both the numerator and denominator before you can check either. They will be checked together. **The check will be on whether the ratio is correct, not whether the individual parts are correct.**
- Make liberal use of parentheses to avoid ambiguity.

Numerator:
Denominator:

Answer:

Number: VT^2kR^2

Denominator: $VT^2kR^2+(1-R)^2$

Step6

Let's first convert R to $\frac{1}{Z}$

$$H = \frac{D_0}{D_i} = \frac{Y}{X} = \frac{VT^2kR^2}{VT^2kR^2 + (1-R)^2} = \frac{VT^2k\frac{1}{Z^2}}{VT^2k\frac{1}{Z^2} + (1-\frac{1}{Z})^2} = \frac{VT^2k}{Z^2 - 2Z + (1+VT^2k)}$$

Take the denominator polynomial and find its roots.

$$p = 1 \pm \mathrm{i}\sqrt{VT^2k}$$

Step7

When k = 1, T = 0.1 second, and V = 0.1 m/s ,

$$p_1 = \frac{2 + \sqrt{(VT^2k)^2 - 4VTk}}{2} = 1 + 0.031623j$$

$$p_2 = \frac{2 - \sqrt{(VT^2k)^2 - 4VTk}}{2} = 1 - 0.031623j$$

We choose p_1, then p_1=$1 + 0.031623j \Rightarrow p_1 \approx e^{0.0316224nj}$

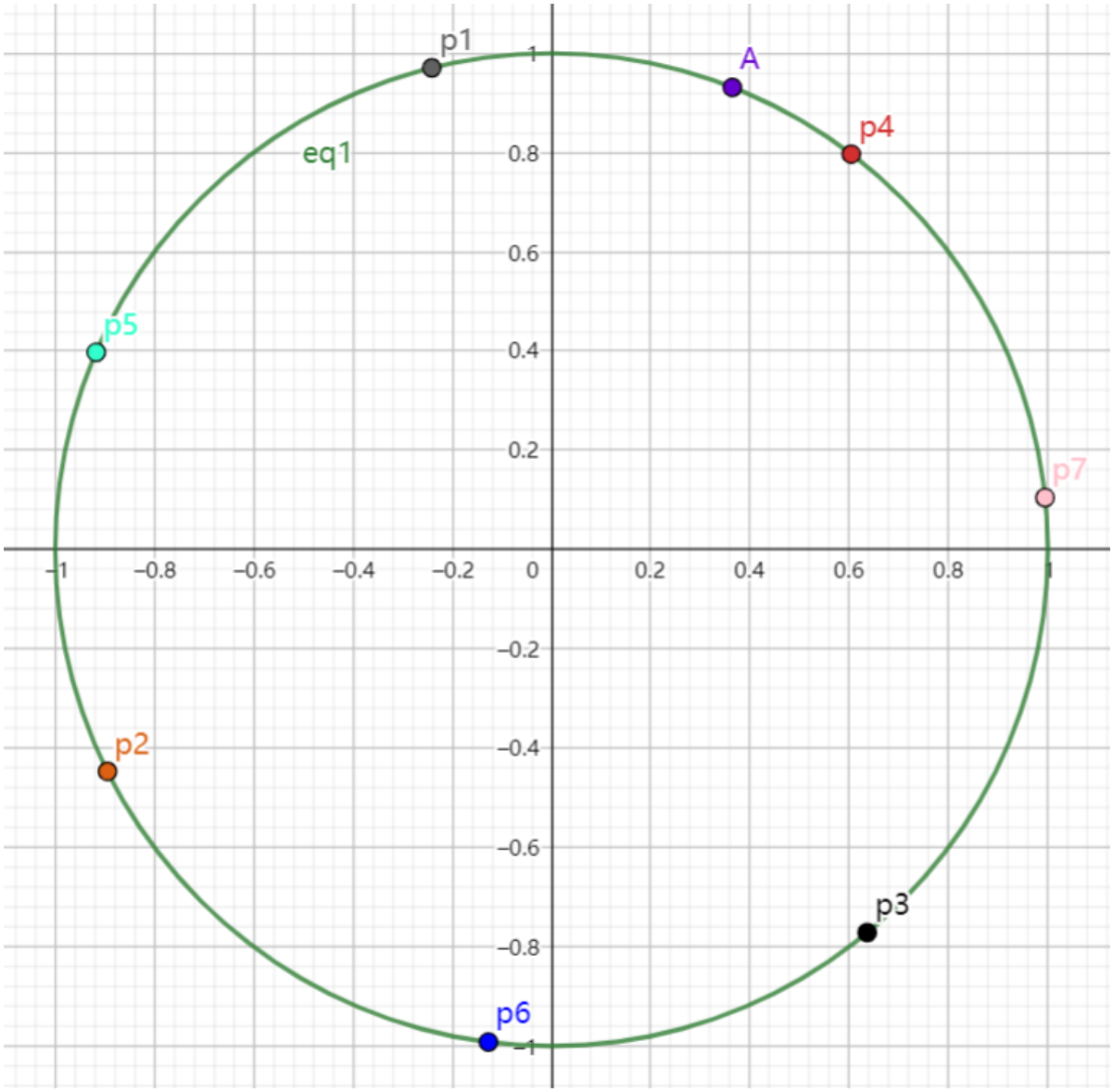
The red line represents the real part, and the green line represents the imaginary part.

It shows that the response of system is periodic, with a period of $\frac{2\pi}{0.0316224}$

Checkoff 2.

Wk5.3.3 In the complex plane, the magnitude of p_1 is 1 and Ω is 0.0316224.

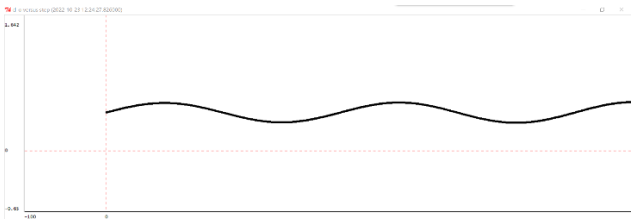
Let's call pn for p to the NTH power. So, the position of the poles in the complex plane is as follows.



Point A refers to p8.

We find that the poles are all on the circle of r equals one. This tells us that the system behavior is always oscillating with period.

We change Gain=75 in propWallFollowBrainSkeleton.py. to Gain=1, and generate vehicle trajectory diagram.

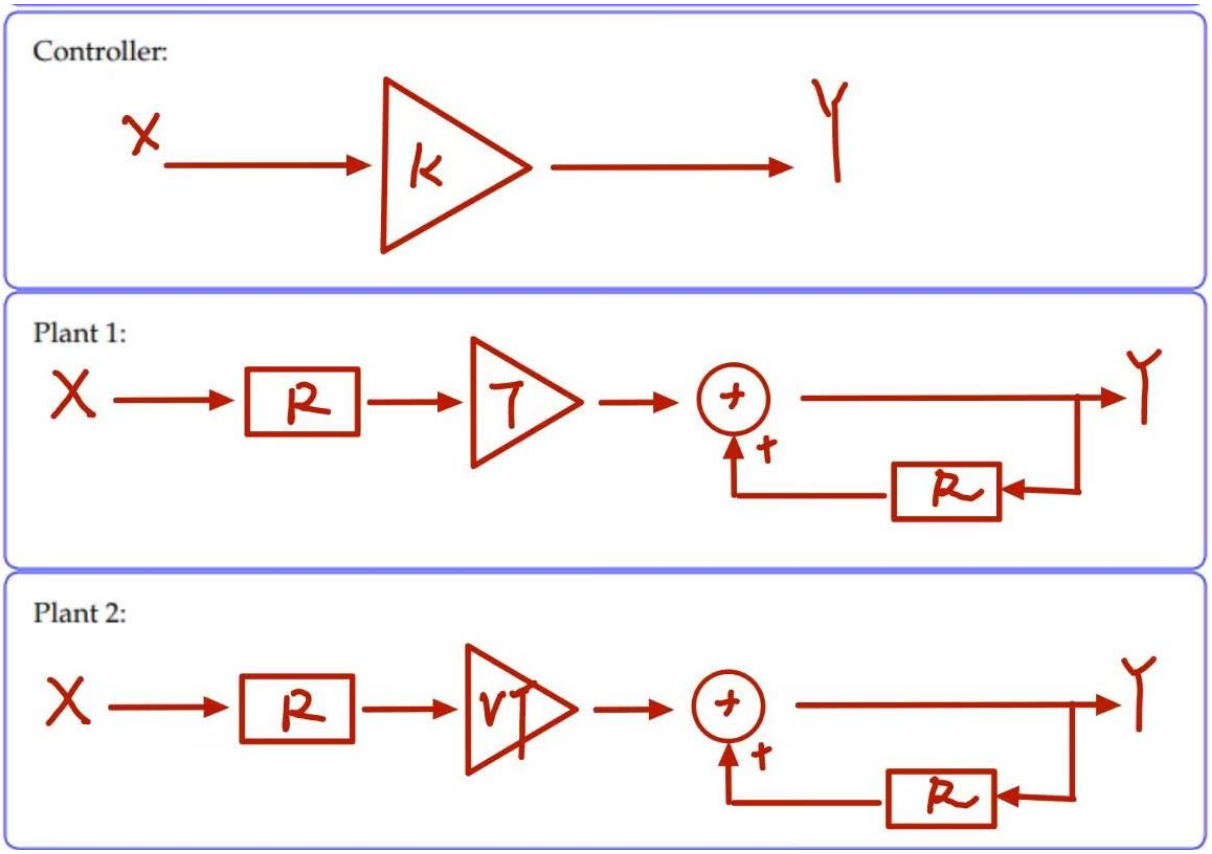


In Checkoff 1, whatever k is, the system is always oscillating periodically. This agrees with the results we obtained in Step 7.

We find that when k=1, the behavior of both systems is oscillating around the expected value.

Since the V of the two is the same and the time interval for the sensor to obtain data is the same, the period of the two is the same at this time

Check Yourself 2. **Use gains, delays, and adders to draw system diagrams representing the controller, plant 1, and plant 2 from the previous section.**



Step8

```
def controller(k):
    return sf.Gain(k)
    pass
def plant1(T):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T)),sf.FeedbackAdd(sf.Gain(1),sf.R()))
    pass
def plant2(T, V):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T*V)),sf.FeedbackAdd(sf.Gain(1),sf.R()))
    pass
```

Step9

```
def wallFollowerModel(k, T, V):
    return sf.FeedbackSubtract(sf.Cascade(sf.Cascade(controller(k),plant1(T)),plant2(T,V)),sf.Gain(1))
    pass
```

My codes for wallFollowerModel.py are as follows.

```
import lib601.sig as sig
import lib601.ts as ts
import lib601.poly as poly
import lib601.sf as sf

def controller(k):
    return sf.Gain(k)
    pass
def plant1(T):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T)),sf.FeedbackAdd(sf.Gain(1),sf.R()))
    pass
def plant2(T, V):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T*V)),sf.FeedbackAdd(sf.Gain(1),sf.R()))
    pass
def wallFollowerModel(k, T, V):
    return sf.FeedbackSubtract(sf.Cascade(sf.Cascade(controller(k),plant1(T)),plant2(T,V)),sf.Gain(1))
    pass
```

Wk5.3.4

```
import math
def controller(k):
    return sf.Gain(k)
    pass
def plant1(T):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T)),sf.FeedbackAdd(sf.Gain(1),sf.
    pass
def plant2(T, V):
    return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T*V)),sf.FeedbackAdd(sf.Gain(1),s
    pass
def wallFollowerModel(k, T, V):
    return sf.FeedbackSubtract(sf.Cascade(sf.Cascade(controller(k),plant1(T)),pla
    pass
##This is Wk5.3.4##
##def wallFollowerModel(k, T, V):
##
##    return sf.FeedbackSubtract(sf.Cascade(sf.Cascade(sf.Gain(k),sf.Cascade(sf.C
##
##    pass
a= wallFollowerModel(1,0.1,0.1).dominantPole()
print(a)
b=polar(a)
print(b)
period=2*math.pi/b[1]
print(period)
```

If you want to run this code, you need to first comment the wallFollowerModel code above.

Step10

According to the formula:

$$p = 1 \pm \mathrm{i}\sqrt{VT^2k}$$

We get:

$$p = \sqrt{1 + kVT^2}e^{\pm j\Omega n},\ \Omega = arctan\sqrt{kVT^2}, period = \frac{2\pi}{\Omega} = \frac{2\pi}{arctan\sqrt{kVT^2}}$$

So, we use the following code to do this:

```
import lib601.sig as sig
import lib601.ts as ts
import lib601.poly as poly
import lib601.sf as sf

from cmath import *
import math

def controller(k):

   return sf.Gain(k)

   pass

def plant1(T):

   return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T)),sf.FeedbackAdd(sf.Gain(1),sf.R()))

   pass

def plant2(T, V):

   return sf.Cascade(sf.Cascade(sf.R(),sf.Gain(T*V)),sf.FeedbackAdd(sf.Gain(1),sf.R()))

   pass

def wallFollowerModel(k, T, V):

   return sf.FeedbackSubtract(sf.Cascade(sf.Cascade(controller(k),plant1(T)),plant2(T,V)),sf.Gain(1))

   pass

a= wallFollowerModel(1,0.1,0.1).dominantPole()
print(a)
b=polar(a)
print(b)
period=2*math.pi/b[1]
print(period)
```

k	period
0.05	888.591
0.5	281.039
5	89.006
50	28.561
75	23.505
100	20.515
110	19.620

Checkoff 3

*Wk5.3.5***Show your results to a staff member. Explain similarities and differences between your results in this part and your results**

in Checkoff 1.

Compared with Checkoff1, we find that when the value of k increases, the period of both parts decreases.

In fact, neither of them will eventually converge to the expected value and oscillate around the expected value. When we simulate a physical system, even if the behavior of the system is found to be oscillating near the expected value by theoretical calculation, the behavior of the system can still be approximated to be convergent within the allowable range of errors.

Summary

*1. When using dominantPole to find the poles of system functions, the final result obtained is the standard form of complex numbers. But what we want more often is an exponential form. Therefore, in designlab.work.py, we introduced cmath and math from the python library to make it easier to convert normal shapes to polar coordinates using the polar function. Meanwhile, to further find the period of the function, we use period=2*math.pi/Ω to solve it.*

2. When we analyze an actual system, the factors to consider are often multifaceted. Even if suitable conditions are found during theoretical analysis, the actual situation still needs to be considered. In step 10 we found that when the value of k increases, the oscillation period of the system behavior is greatly shortened. So, does it mean that we can achieve the effect of convergence of system behavior by increasing k and decreasing the period indefinitely? The answer, of course, is no. When we increase k, although period decreases, the rotational speed of the system behavior increases a lot. When the speed is too large, the safety of the system is difficult to guarantee.

At the same time, even if we do not have an ideal solution in theoretical analysis, as long as it is within the allowable range of errors, we can approximate that the experimental results meet the needs. For example, when k>20 in Checkoff1, we have actually been able to approximate the system behavior as convergence,

and the next theoretical analysis is more inclined to improve the accuracy of the experiment.

3. In checkoff1 we did not consider the case of k<0 because the expected value is always the distance from the right wall. When the input value (that is, the actual distance from the right) is greater than the expected value, it should turn right, and if it is less than the expected value, turn left. If k is negative, the direction of the system behavsior is changed.