

Work for Week 6 (Resubmit)

Software Lab

I've implemented a transfer function as a class file taking in two polynomials A and B, A being the numerator and B being the denominator. To find the natural frequencies, we find the zeroes of the denominator (using the methods found in the `polynomial` class).

Black's formula, after some algebraic manipulation, ends up being

$$\tilde{H}(z) = \frac{\tilde{A}}{\tilde{A} + \tilde{B}}$$

```
import polynomial
from polynomial import *
import math
from math import *
import Numeric
import random
import roots

class TransferFunction:
    def __init__(self, A, B):
#A and B are polynomials
        #store a transfer function as a numerator and a denominator
        self.num = A
        self.denom = B

    def __mul__(self, Y):
        #multiply the respective polynomials together
        return TransferFunction(self.num*Y.num, self.denom*Y.denom)

    def Black(self):
        #do some algebra to get a Black's formula suitable for this program
        return TransferFunction(self.num, self.num + self.denom)

    def naturalFrequencies(self):
        #for some strange reason, it returns the negative of the correct roots
        return self.denom.roots()

Test Case (Using example from the software lab)
>>> A1 = Polynomial([1,1])
>>> B1 = Polynomial([1,2])
>>> A2= Polynomial([2,1])
>>> B2 = Polynomial([1,3])
>>> H1 = TransferFunction(A1, B1)
>>> H2 = TransferFunction(A2, B2)
>>> H3 = H1 * H2
>>> print H3.num
2.00(z^2) + 3.00(z) + 1.00
>>> print H3.denom
1.00(z^2) + 5.00(z) + 6.00
>>> H4 = H3.Black()
>>> H4.naturalFrequencies()
[(-1.3333333333333333+0.7453559924999299j), (-1.3333333333333333-
0.7453559924999299j)]
```

Robot Lab

In lab, we proved that the controller with a single gain was unstable. We did this by doing the following:

$$\begin{aligned}d[n] &= d[n-1] + v \delta t \theta[n] \\ \theta[n] &= \theta[n-1] + \delta t K e[n-1]\end{aligned}$$

We can then substitute and apply the Z-transform to both sides

$$\begin{aligned}\tilde{\Theta}(z) &= z^{-1} \tilde{\Theta}(z) + \delta t K z^{-1} \tilde{E}(z) \\ \tilde{D}(z) &= z^{-1} \tilde{D}(z) + v \delta t z^{-1} \tilde{\Theta}(z)\end{aligned}$$

After some algebraic manipulation and substituting tilde E(z), we obtain

$$\frac{\tilde{D}(z)}{\tilde{E}(z)} = \frac{v \delta t^2 K z^{-2}}{(1 - z^{-1})^2}$$

If we substitute $\tilde{D}_d(z)$ by solving $\tilde{E}(z)$ we get a transfer function that relates $\tilde{D}(z)$ with $\tilde{D}_d(z)$

$$\begin{aligned}\tilde{H}(z) &= \frac{\tilde{D}(z)}{\tilde{D}_d(z)} \\ &= \frac{v \delta t^2 K z^{-2}}{1 - 2z^{-1} + z^{-2}(1 + v \delta t^2 K)}\end{aligned}$$

The natural frequencies of \tilde{H} are complex and greater than zero; thus, the system is oscillatory and unstable.

Simple Controller Transfer Function Definition and Interaction

```
#Simple Controller Transfer Function

dt = 0.2
K = -2.0
V = 0.1

X = Polynomial([1, -2, 1])
D = Polynomial([(dt**2)*V])

H = TransferFunction(X, D)
print "H"
print H.num
print H.denom

Dd = H.Black()
print "Dd"
print Dd.num
print Dd.denom

Ans = Dd.naturalFrequencies()
print Ans

>>>
H
1.00 (z^2) + -2.00 (z) + 1.00
0.00
Dd
1.00 (z^2) + -2.00 (z) + 1.00
1.00 (z^2) + -2.00 (z) + 1.00
[(1-0.06324555320336761j), (1+0.06324555320336761j)]
>>>
```

As expected, these natural frequencies are complex, and have a magnitude > 1 .

The double-gain controller

From the lab handout:

$$\theta[n+1] = \theta[n] + \delta t (K_1 e[n] + K_2 e[n-1])$$

Our goal is to find values for K_1 and K_2 such that the robot will be stable.

With the same kind of treatment as the single gain controller, we were able to find that

$$H'(z) = \frac{v \delta t^2 (K_1 z^{-2} + K_2 z^{-3})}{z^3 - 2z^2 + z(1 + v \delta t^2 K_1) + K_2 v \delta t^2}$$

Now, to find the poles for this beast...

At Bill's suggestion, I noted that a polynomial of degree 3 must have 3 roots of the form

$$(z - r_1)(z - r_2)(z - r_3) = 0$$

If we multiply this out, we get the following polynomial:

$$z^3 + (-r_1 - r_2 - r_3)z^2 + (r_2 r_3 + r_1 r_3 + r_1 r_2)z - r_1 r_2 r_3 = 0$$

If we set this equal to the denominator of the transfer function, we can use the method of undetermined coefficients to obtain our natural frequencies.

If we let $v \delta t^2$ equal 1 (for simplicity), we get that

$$K_1 = r_2 r_3 + r_1 r_3 + r_1 r_2 - 1$$

$$K_2 = -r_1 r_2 r_3$$

$$r_1 + r_2 + r_3 = 2$$

One solution that satisfies this is $r_1 = r_2 = r_3 = \frac{2}{3}$

$$K_1 = \frac{1}{3}$$

$$K_2 = 0.3$$

With all the negative exponents, I'm not sure how to use this with my transfer function class.

Comparison of the two robot controllers

The single-gain controller was supposed to, in theory, eventually oscillate out of control; in a few cases it did this (but this might be due to sonar signals bouncing off of the wall and not returning), but most of the time the robot was happy to swing back and forth around the center.

The double-gain controller did a better job of not oscillating too much, but there was still some noticeable swinging around (though there was no more robot-to-wall contact). Perhaps if we had tweaked our gains a bit more we would have been able to center the robot perfectly?\

Control Code

##Week 6 Motor Control

```
##Feedback things - New controller
```

```
class Zrobot:
    def step(self):
        self.ranges()
        motorOutputs(0.1, erf())

    def setup(self):
        print "Starting!"
        self.desired = 20

    def ranges(self):
        self.distances = sonarDistances()           #Here, I'm defining which groups
of sensors will be used
        self.left = min(distances[0:2])
        self.right = min(distances[6:8])
        self.front = min(distances[3:5])
        #print 'RotVel = ', rotVel                   #Debug tools
        #print 'Speed = ', linVel
        print 'SONAR'
        print 'Left Min = ', self.left
        print 'Right Min =', self.right
        print 'Front Min =', self.front
        #print 'Difference = ', difference

    def erf(self):           #error function
        nerf = (0.5)*k*(self.desired - self.left) + (0.5)*K*(self.previousError())
```

k and K are our two gains.

The single gain controller is similar, except that nerf is the single-gain formula instead of the double gain.