

Midterm 1 Practice Solutions

1 Robot behavior

1.

```
class TBTurnSmoothly(SM):
    def getNextValues(self, state, inp):
        currentTheta = inp.odometry.theta
        return (state,
                io.Action(rvel = self.rotationalGain * (thetaDesired -
                                                            currentTheta),
                           fvel = 0))
```

2. The difference equations are:

$$\Theta[n] = \Theta[n-1] + T\Omega[n-1]$$

$$\Omega[n] = K(\Theta_{des}[n-1] - \Theta[n-1])$$

The operator version is:

$$(1 - \mathcal{R})\Theta = T\mathcal{R}\Omega$$

$$\Omega = K\mathcal{R}(\Theta_{des} - \Theta)$$

Combining we get:

$$1 - \mathcal{R}\Theta + KTR\mathcal{R}^2\Theta = KTR\mathcal{R}^2\Theta_{des}$$

We accepted a number of variations.

3. The system function is:

$$\frac{\Theta}{\Theta_{des}} = \frac{KTR\mathcal{R}^2}{1 - \mathcal{R}\Theta + KTR\mathcal{R}^2}$$

Substitute $\mathcal{R} = 1/z$ in the denominator polynomial and we get:

$$z^2 - z + KT$$

The roots of this polynomial are the poles:

$$\frac{1 \pm \sqrt{1 - 4KT}}{2}$$

Note that for $K = 0$, one of the poles is 1.0. For $K < 0$, one of the poles is always greater than 1.0. This makes sense since for $K < 0$, the feedback increases the error.

For $T = 0.2$, for $K = 5$, we have a pole of magnitude 1.0 and they get bigger with bigger gain. The range of stable K is $0 < K < 5$

In general, when $4KT > 1$ we have complex poles. For monotonic convergence we need real poles, so we want:

$$0 < K \leq 1/(4T)$$

Note that when $K = 1/(4T)$, we have the lowest magnitude pole, the pole approaches 1 as K decreases towards 0, so:

For $T = 0.2$, the best value of K is 1.25, which leads to a pole of $1/2$.

4. The analysis above shows that when T increases the range of stable gains decreases. Basically the effective gain is KT . As the T increases we need smaller and smaller gains to keep monotonic convergence. The result is very sluggish performance.
- 5.

$$\Theta[n] = \Theta[n-1] + T\Omega[n-1]$$

$$\Omega[n] = \Omega[n-1] + T\Xi[n-1]$$

$$\Xi[n] = K_1 E[n] + K_2 E[n-1]$$

$$E[n] = \Theta_{des}[n] - \Theta[n]$$

where Ξ is the acceleration. So, basically, the velocity is the integrated acceleration and the position is the integrated velocity. And, the acceleration is given by the gains times the position errors.

```
def robot(k1, k2):
    dt = 0.2
    control = sf.SystemFunction(poly.Polynomial([k2, k1]),
                                poly.Polynomial([1]))
    vel = sf.SystemFunction(poly.Polynomial([T, 0]),
                            poly.Polynomial([-1, 1]))
    pos = sf.SystemFunction(poly.Polynomial([T, 0]),
                            poly.Polynomial([-1, 1]))
    rob = sf.FeedbackSubtract(sf.Cascade(control,
                                          sf.Cascade(vel, pos)))
    return abs(rob.dominantPole())
```

Depending on the resolution of the search, one gets very different answers.

```

optOverGrid(robot, -10, 10, -10, 10, 1, 1)
(0.99999999999999956, (1, -1))
optOverGrid(robot, -10, 10, -10, 10, 0.5, 0.5)
(0.84617988641100905, (2.5, -2.0))
optOverGrid(robot, -10, 10, -10, 10, 0.25, 0.25)
(0.74999999999999933, (1.75, -1.5))
optOverGrid(robot, -10, 10, -10, 10, 0.1, 0.1)
(0.68863445766586584, (1.4999999999999816, -1.3000000000000189))

```

2 Weighted Objective

1.

There are many ways of writing this. Here are a few.

```

def argmax(elements, f):
    bestScore = None
    bestElement = None
    for e in elements:
        score = f(e)
        if bestScore == None or score > bestScore:
            bestScore = score
            bestElement = e
    return bestElement

```

```

def argmax(elements, f):
    bestElement = elements[0]
    for e in elements:
        if f(e) > f(bestElement):
            bestElement = e
    return bestElement

```

```

def argmax(elements, f):
    vals = [f(e) for e in elements]
    return elements[vals.index(max(vals))]

```

```

def argmax(elements, f):
    return max(elements, key=f)

```

2.

Here are a couple that work:

```
WOPQ([Fish.length, Fish.width], [0.9, 0.1])
WOPQ([lambda x: x.length(), lambda x: x.width()], [0.9, 0.1])
```

3.

```
def extract(self):
    def priority(e):
        return sum([w*f(e) for (w,f) in zip(self.weights,self.features)])
    best = argmax(self.elements, priority)
    self.elements.remove(best)
    return best
```

4.

```
wopq.extract() = Sydney
```

Sydney is the longest fish and the priority weights heavily favor length.

5.

```
class WOPQ(FPQ):
    def __init__(self, features, weights):
        def priority(e):
            return sum([w*f(e) for (w,f) in zip(weights,features)])
        FPQ.__init__(self, priority)
```

That's it... the other methods are provided by the FPQ class.

6.

```
def featureRange(feature, elements):
    vals = [feature(e) for e in elements]
    return max(vals) - min(vals)

def selectFeatures(features, n, elements):
    return sorted(features, key = featureRange)[-n : ]
```

3 More Oop

You are told that we have a class A, with one initialization argument, with a method f that takes one argument and returns a number.

```
>>> x = A(2)
```

```
>>> x.f(3)
```

```
9
```

Define a class B that is a subclass of class A. Class B has the same initialization as A and a method `f` that returns twice whatever value the `f` method of A would return. You don't need to know what the code for A is.

```
>>> x = B(2)
>>> x.f(3)
18
```

```
class B(A):
    def f(self, x):
        return A.f(self, x)*2
```

2 points for correct answer (no `__init__`, passing self)

1 point for minor error

4 In the Tank

Imagine a cylindrical water tank with a source of water entering the bottom of the tank and a sensor to measure the depth of the water. The source of water can be adjusted so that water enters (or exits) the tank at a desired rate. In addition to this controlled source of water, there is also a leak in the tank, through which water flows regardless of the controlled flow. The diameter of the tank is such that one inch in depth is one gallon of water. The following variables characterize this problem:

- $d[t]$ is the depth, in inches, of the water in the tank at time t .
- $v[t]$ is the volume of water that flows through the controlled source of water in the time interval $[t, t + 1]$.
- $o[t]$ is the volume of water that leaks from the tank in the time interval $[t, t + 1]$. Assume $o[t]$ is proportional to $d[t]$, with a proportionality constant of 0.1
- $g[t]$ is the desired (goal) depth of water (in inches) in the tank at time t
- $s[t]$ is the sensed depth (in inches) of water in the tank at time t ; assume the sensor introduces one unit of delay, so that $s[t] = d[t - 1]$

a. Write a difference equation that describes the depth of water in the tank ($d[\cdot]$) in terms of the flow through the controlled source ($v[\cdot]$).

$$d[t + 1] = d[t] + v[t] - o[t]$$

$$o[t] = 0.1d[t]$$

$$d[t + 1] - 0.9d[t] = v[t]$$

3 points if correct

2 points if expression contains $o[t]$ but otherwise correct

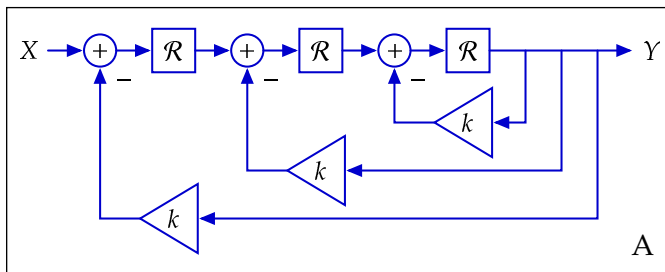
b. Write a difference equation describing a proportional controller that specifies $v[\cdot]$ as a function of sensed depth $s[\cdot]$ and desired depth $g[\cdot]$. Let α represent the constant of proportionality.

$$v[t] = \alpha(g[t] - s[t])$$

3 points if correct

5 Linear Systems (20 points)

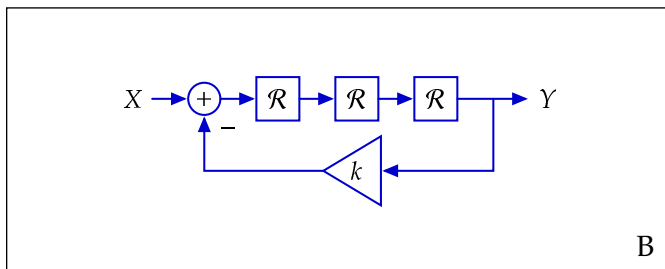
Part a. Several systems are illustrated below. The left column shows block diagrams. The right column shows system functions and pole locations for $k = 0.9$. Indicate which panel on the right (if any) corresponds to each panel on the left by drawing a straight line from A, B, C, and/or D to its partner. [Your answer should include 4 or fewer straight lines.]



$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k\mathcal{R}^3 + k\mathcal{R}^2 + k\mathcal{R} + 1}$$

poles: 1.72; $-0.41 \pm 0.59j$

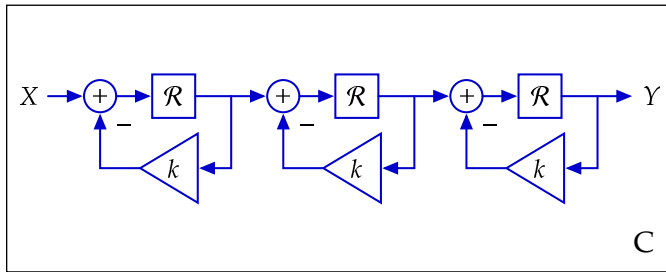
E



$$\frac{Y}{X} = \frac{1}{k\mathcal{R}^3 + 1}$$

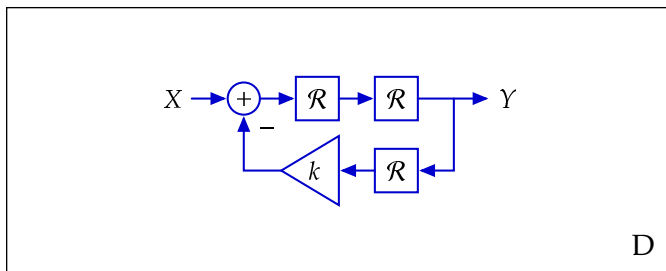
poles: $-0.48 \pm 0.84j$; 0.96

F



$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k\mathcal{R}^3 + 1}$$

poles: $-0.48 \pm 0.84j$; 0.96



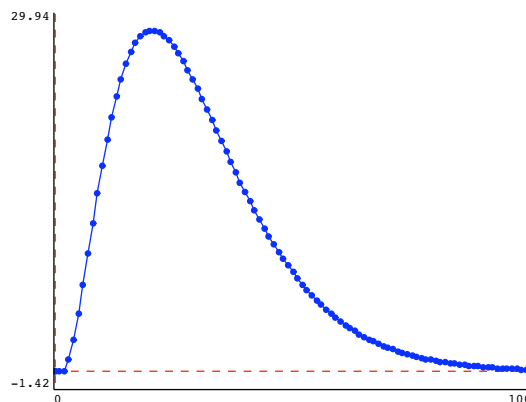
$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k^3\mathcal{R}^3 + 3k^2\mathcal{R}^2 + 3k\mathcal{R} + 1}$$

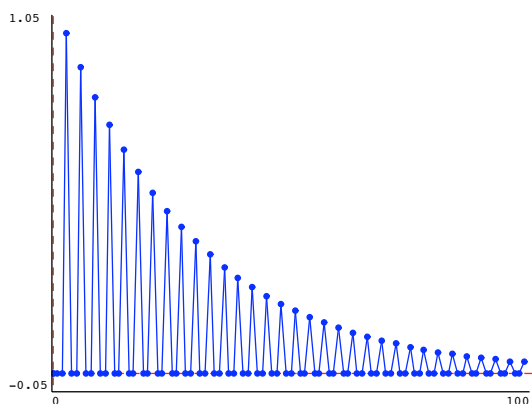
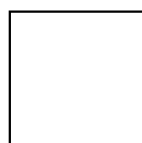
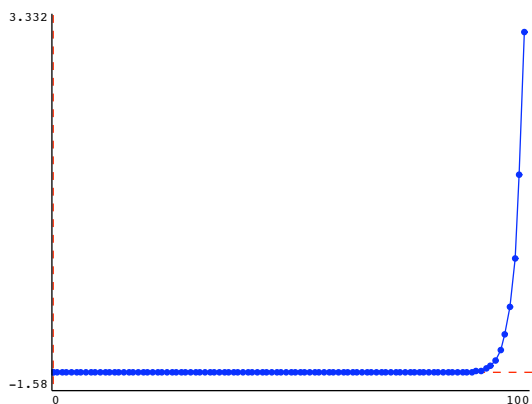
poles: 0.9; 0.9; 0.9

- A → E → +3 points if correct
- B → G → +3 points if correct
- C → H → +3 points if correct
- D → nowhere → +2 points if correct

Part b. Unit sample responses. Indicate which (if any) of the systems on the preceding page could have generated the following unit sample responses (with the system initially at rest) by entering A, B, C, D, E, F, G, H or “none” in the corresponding boxes.

Put exactly one answer in each box. You will receive full credit if your answer is correct regardless of whether other answers are also correct.





top: C or H → +3 points

middle: A or E → +3 points

bottom: D → +3 points B, F, G → +2 points