MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.01—Introduction to EECS I
Spring Semester, 2008

Sample Midterm Exam—Distributed March 16

Ground Rules

This is a sample midterm to help you study and to give you a sense of what kinds of problems to expect on the real midterm. You can pick up the midterm in 34-501 (the 6.01 lab) on Wednesday, May 19, any time between 8:30AM and 7:00PM, and return it within 3 hours after you pick it up.

If you plan to print out your answers (good idea) we suggest that you work out the mechanics of how you are going to print *before* you pick up the exam. "My dog ate my printer" is not a valid excuse for being late.

When you pick the exam up, you'll see the following instructions:

- Read through this exam and ask any questions you have before you leave. If you find that you have a question after that, try to pick your own answer, and write down what assumption you're making. We will not answer your questions after you've left the room.
- Turn the exam in to room 34-501 (the 6.01 lab) *at most three hours* after you pick it up. You may turn in your solution early, but late papers will be penalized 10% of the grade value for every ten minutes you are late.
- For the problems that ask you to write Python code, you should just think it through and write down the code. **You must not** try to actually debug it (it will take too long). Don't spend time trying to be sure the syntax is correct. It's more important to communicate your ability to think algorithmically than to worry about the fine-grained details of syntax. Be sure to add comments that explain what you're trying to do.
- We expect you to spend less than an hour per question. The extra time is to let you relax and think.
- Your answer may be handwritten or typed; it should be legible in either case. We reserve the right to refuse to grade illegible papers.
- Label your answers clearly, we won't look through long printouts for something that looks like an answer.
- You may read anything you like (labs, books, the web) during the exam, but you may not communicate with anyone else.
- When you turn in your paper, the sheets *must* be stapled together and you *must* have your name and section number on each sheet. Papers that do not do this will *not* be graded and you will *not* get credit for the exam.

Programming

Soar, as you recall, contains a procedure `sonarDistances()`, which returns the list of distances reported by the robot sonars. For this problem, assume that there is only one sonar, and that the procedure `readSonar()` returns that sonar reading as a single number. You've undoubtedly noticed that the robot sonars sometimes give flaky readings. So it might be useful to have a procedure like `readSonar`, but which returns the entire history of sonar readings, or the average of the sonar readings over several calls, or the minimum and maximum readings. In this problem, we'll ask you to implement such a procedure. Please use only the basic functions built into Python (e.g., don't use fancy packages you might find on the Web).

We'll start with the following class:

```
class RangeTracker:
    def __init__(self):
        self.saved = []
    def values(self):
        self.saved.append(readSonar())
        return self.saved
```

The `RangeTracker` class behaves as follows (assuming that the first few sonar readings are 10, 9, 11, and 12).

```
>>> rt = RangeTracker()
>>> rt.values()
[10]
>>> rt.values()
[10, 9]
>>> rt.values()
[10, 9, 11]
>>> rt.values()
[10, 9, 11, 12]
```

Question 1: Add methods to the `RangeTracker` class to return the maximum and minimum of the saved values (one method for each).

Question 2: The mean, or average, of a set of N numbers $x_1, \ldots, x_N$ is the sum of the x's over the number of x's:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \ .$$

Add a method to the `RangeTracker` class that returns the mean of the sonar readings.

Question 3: The standard deviation of a set of numbers is a measure of how spread out the numbers are around the mean. It can be computed as

$$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \ .$$

Add a method to the `RangeTracker` class that returns the standard deviation of the sonar readings.

Question 4: Add a method to the `RangeTracker` class that takes a function `f` as an argument, and returns the mean of the values of `f` applied to each of the `x` values. That is:

$$\frac{1}{N}\sum_{i=1}^{N} f(x_i) \ .$$

Question 5: The above parts of the problem asked you to add new methods to the `RangeTracker` class. An alternative way to add new methods—without modifying the original class—is to use inheritance. Define a new class called `TrackerWithReset`, a subclass of `RangeTracker`, that has a `reset` method that reinitializes the list of saved values to the empty list.

In assessing your performance on this programming problem, we're interested not only in whether your programs return the right answers, but in whether your code is and shows good style. The definition of "good style" is somewhat idiosyncratic, but for us, it includes things like

- Do you reuse pieces of code to achieve compactness and clarity (for example, does your code for computing the standard deviation make use of your code for computing the mean)?
- Do you take advantage of list comprehension, rather than writing explicit for or while loops?

It is not necessary for you to test and debug your code. We won't be taking off points for trivial syntax errors. Similarly, on the final exam, we may ask you to write code, but we certainly won't expect you to run and debug it.

Helpful hint: The mean of the set of numbers 1, 2, 3, . . . , 10 is 5.5, and the standard deviation is 3.02766 .

Feedback control

In lab for week 6, you programmed the robot to drive down the center of a narrow corridor, maintaining a desired distance desired from the center. You used a control algorithm based on the error

$$e[n] = d_{desired}[n] - d[n] \ ,$$

where $d[n]$ is measured distance to the left of the center. You modeled the robot's dynamics by

$$\begin{aligned} d[n] &= d[n-1] + V\delta_T\theta[n-1] \\ \theta[n] &= \theta[n-1] + \delta_T\Omega[n-1] \end{aligned}$$

where $\theta[n]$ is the angle of the robot heading, $\Omega[n]$ is the robot's angular velocity, V is the robot's forward speed, and δ_T is the time step. You then implemented a control law that produced values for $\Omega[n]$.

Suppose $\Omega[n]$ is specified according to the control law

$$\Omega[n] = Ke[n-2] \ ,$$

that is, Ω is proportional to the error two time-steps ago.

Question 6: What is the system function for the whole system (which takes $d_{Desired}$ as input and generates d as output)?

Question 7: 2. Suppose we choose K so that $KV\delta_T^2 = 1$. What are the poles? Is the system stable? How about for $KV\delta_T^2 = -1$?

Note: The Python root finder we used in class can compute roots of polynomials only up to third order. There are some simple root finders on the Web, for example, at http://xrjunque.nom.es/precis/rootfinder.aspx

Terminating State Machines

Write a terminating state machine class that inherits from the `TBDrive` class that you wrote in PS 4 and which does the same thing, that is, move the robot a specified distance. However, it should use the action `goFast` for the first 80% of the displacement and then use the action `go` for the rest. Make your new class as simple as possible.

Solution to programming problem

The point of this problem is to see if you can define a class, write simple methods, and use inheritance. We also want to know if you've learned to use list comprehension and simple functional programming.

While the answer code we've provided here actually runs, we strongly suggest that in doing the exam you *do not attempt to debug your code.* You probably won't have time for that during the exam, and we won't be taking off points for minor syntactic errors. Showing that you understand the basic constructs, and using good style, will be more a important use of your time than rooting out minor bugs in your code.

Question 1 The two new methods are

```
def maxval(self):
# to return the maximum of the saved values
# we simply apply the max function to the list
   return max(self.saved)

def minval(self):
# similar to max
    return min(self.saved)
```

Note how we used the fact that `min` and `max` can take the list of a saved items directly, so that we did *not* write a loop. This is an echo of the functional style that is preferred to explicit loops. Note also that the instance variable is referenced as `self.saved`, not `saved`.

Question 2 The code here is

```
def mean(self):
# the average is the sum of the values divide by the
# number of values
    return sum(self.saved) / float(len(self.saved))
```

We might also have used functional style here, with `add` and `reduce`

```
    return reduce(add,self.saved)/float(len(self.saved))
```

Note that in order for this to run, you'd have to import `add` from Python's `operator` package. In grading, we might penalize people for solving this problem with a `while` or `for` loop, even if their program gave the correct answer. Observe the unfortunate need to convert the denominator to a float.

Question 3 The code here is:

```
def stdev(self):
    # compute the average of the values and the number of values
    m = self.mean()
    n = len(self.saved)
    # use map/reduce to compute the sum of the squared differences
    sumsq = sum([(x - m)**2 for x in self.saved])
    return ((1.0/(n-1))*sumsq)**0.5
```

We could have written this all in one line, but we defined local variables m for the mean of the readings and n for the number of readings, to make the code more readable. Readability is always a good thing to keep in mind when you are writing something for graders, or co-workers, or even yourself, a few days later, to read. Also, by defining a local variable for the mean, we avoided re-computing it n times. We used a map/reduce form (expressed as a list comprehension, and the built-in `sum`) to compute the sum of the squares.

Question 4 Here the code is

```
    def meanVals(self, f):
        # use map/reduce to find the sum and then divide to get the mean
        return sum([f(x) for x in self.saved])/float(len(self.saved))
```

We're looking to see that you recognize that you can pass a procedure `f` as an argument. Otherwise, this is identical to question 2.

Question 5 This is a simple use of inheritance:

```
class TrackerWithReset(RangeTracker):
    # a TrackerWithReset is a RangeTracker with an additional reset method
    def reset(self):
        self.saved = []
```

Solution to feedback problem

This problem is very similar to what you've been doing in lab these past two weeks. The lesson you should take away from this in studying is that there will be problems closely related to one or more of the labs.

Question 6: If we use Black's formula, then the analysis follows the one presented in lecture on March 11 and in the course notes. If we let H_{robot} be the system that relates the distance d to the turning rate Ω, we have

$$\begin{aligned} H_{\text{robot}} = \frac{d}{\Omega} &= \frac{d}{\theta} \cdot \frac{\theta}{\Omega} \\ &= \frac{V\delta_T \mathcal{R}}{1-\mathcal{R}} \cdot \frac{\delta_T \mathcal{R}}{1-\mathcal{R}} \\ &= \frac{(\delta_T)^2 V \mathcal{R}^2}{(1-\mathcal{R})^2} \quad . \end{aligned}$$

Then for the overall system we have by Black's formula:

$$\frac{d}{D_{\text{desired}}} = \frac{H_{\text{control}} H_{\text{robot}}}{1 + H_{\text{control}} H_{\text{robot}}}$$

The problem now specifies that

$$\Omega[n] = Ke[n-2]$$

i.e.,

$$H_{\text{control}} = K\mathcal{R}^2$$

So putting this together, gives the overall system function as

$$\begin{aligned} &= \frac{K\mathcal{R}^2 \cdot \frac{(\delta_T)^2 V\mathcal{R}^2}{(1-\mathcal{R})^2}}{1 + K\mathcal{R}^2 \cdot \frac{(\delta_T)^2 V\mathcal{R}^2}{(1-\mathcal{R})^2}} \\ &= \frac{K\,(\delta_T)^2\, V\mathcal{R}^4}{1 - 2\mathcal{R} + \mathcal{R}^2 + K\,(\delta_T)^2\, V\mathcal{R}^4} \quad . \end{aligned}$$

Question 7: The poles are the roots of the polynomial we get when we substitute $z = 1/\mathcal{R}$:

$$1 - 2\left(\frac{1}{z}\right) + \left(\frac{1}{z}\right)^2 + K\delta_T^2 V \left(\frac{1}{z}\right)^4$$

and multiply through by z^4:

$$z^4 - 2z^3 + z^2 + KV\delta_T^2$$

For $KV\delta_T^2 = 1$ the roots of

$$z^4 - 2z^3 + z^2 + 1$$

are two complex pairs $p = -0.3 \pm 0.625j$ and $p = 1.3 \pm 0.625j$ and the system is unstable since the second pair has magnitude greater than 1.

If $KV\delta_T^2 = -1$ the roots of

$$z^4 - 2z^3 + z^2 - 1$$

are two real roots $p = -0.618$ and $p = 1.618$ and the complex pair $p = 0.5 \pm 0.866j$. Here, too, there is a root with magnitude greater than 1, and the system is unstable.

Solution to Terminating State Machine problem

```
class TBDriveFast (TBDrive):
    def currentOutput(self):
        if self.done():
            return stop
        else:
            if poseDist(self.startingPose, self.currentPose) < 0.8*self.d:
                return goFast
            else:
                return go
```