

1 Preliminaries

This programming assignment, like the ones to follow, involves a series of programming tasks. For this assignment and all future problem sets, you are asked to turn in an electronic version of your procedures. Before the due-date of the problem set, copy your files into the directory
`/usr/local/courses/cse/cse233.01/dropbox/ps6/your-group-id`
where `your-group-id` is the person's name in your homework group which you will use for the rest of the semester. Also, for some of the exercises, you are asked to turn in transcripts of your SCHEME session showing the implementation of the procedures you write. Make sure your printed output has both the input to, and output from, the SCHEME interpreter. **You will help the graders out immensely – and increase your chances for partial credit – if you liberally comment your code.**

Reading assignment: Chapter 3, Section 3.5, `ps7-owl.scm`

2 Lab Work – Streams and Delayed Evaluation

In contrast with the standard tools introduced so far in this course, streams provide a different way of reasoning about procedures and processes. In particular, streams are useful for modeling infinite mathematical objects, such as a sequence of samples of a continuous-time signal or the sequence of coefficients of an infinite power series. In this problem set, we will concern ourselves mainly with the former, that is, modelling signals using streams.

Copy `ps7-owl.scm` to your directory to use. You will not need to modify most of the code, although a few templates for use in some of the problems are included at the end.

This problem set concentrates on the use of higher order procedures such as `map`, `filter`, and some others that you must write. You should be familiar with the attached code before starting to work in the lab. There are no large programs to write, but lots of mind-stretching ideas to understand. You will notice that there are lots of pieces to this problem set. Fortunately, most of them involve solutions with small amounts of work that tend to build incrementally. We therefore encourage you to start working on it early so there is time to unstick you if you get stuck.

3 The Amazing Owl

Louis Reasoner has recently been reading about the neurophysiology and psychophysics of the owl's perceptual system and has become fascinated with the owl's uncanny ability to locate prey.

For example, the owl actually has finer visual acuity than humans. Surprisingly, the actual packing of photoreceptors in owl's retina is roughly the same as in human retinas (which may well be the limiting case), but the owl obtains its better acuity by having a bigger eyeball, so that the center of the visual image is nearly twice as large as that of a human. Moreover, since the owl has to locate small prey (e.g. mice) at large distances, it needs some means of capturing 3D information about the world. In part, it does this with an amazing stereo vision system (that is, it uses the differences in the appearance of the world in the two eyes to recover the distance to points in the world). Humans use their ability to change the angle of gaze of their eyes to fixate on objects at different distances in front of them. Owls, on the other hand, have their eyes fixed in their heads and use a hardwired retina to do stereo vision. Specifically, receptors at the top of the two retinas are "wired together" to detect objects at roughly the distance of the owls' feet and, as you move down the retina, the receptors in the two eyes are "wired together" to detect objects at varying distances lying on a slanted plane that extends to the horizon. As a consequence, if the owl wants to figure out how far away something is, it must bring those photoreceptors tuned to different distances into alignment with the object. It does this by nodding its head. Hence, while you may think the owl is looking very wise by doing this, it is actually just trying to figure out how far away you are.

Clearly, the owl needs this incredible visual system for detecting and catching small prey. But surprisingly, it can also locate prey this way at night, in part because of the extreme sensitivity of its visual system, but also because it

also has an incredible auditory system. For example, the owl can detect differences in time of arrival of a sound at its two ears of less than 10 microseconds (and the actual sensitivity may be even smaller). This allows it to determine the orientation of the sound source with respect to itself to an accuracy of less than one degree of arc. This enables it to locate the orientation of some prey in a plane parallel to the ground. To determine how high or low the prey is relative to that plane, the owl needs some other information. In fact, its ears are not symmetrically placed on its head, but are canted relative to one another, so that it can use differences in the perceived sounds to tell the elevation of the source.

Louis is fascinated by this system (as obviously is the author of this problem set) so he decides this would be an interesting thing to simulate. Since he also recently read about streams in *Strife and Interminability of Computer Programming*, he decides to use this idea in building a simulator.

If the prey is located at some distance d from the center of the owl's head, at an angle θ from the straight-ahead direction, if the distance from each ear to the center of the owl's head is a baseline b , and if the speed of travel of an auditory signal is v , then the difference in distance that a sound must travel to reach the left and right ears is given by

$$v(t_l - t_r)$$

where t_l and t_r are the lengths of time it takes the sound to reach the left and right ears respectively. Of course, we don't know when the sound left the prey, but we can in principle measure the difference in arrival of the sound at each ear, $t_l - t_r$.

Some geometric and algebraic manipulations show that as long as $d \gg b$, then the orientation of the prey, θ , is approximately given by

$$\tan \theta \approx \frac{v(t_l - t_r)}{\sqrt{4b^2 - v^2(t_l - t_r)^2}}.$$

If we know the baseline b and the speed of propagation v , and we measure $t_l - t_r$, then we could recover the direction of the prey.

To get an estimate of the distance to the prey, we will assume that sound diminishes according to an inverse-square law. That is, if S is the volume of the sound emitted by the prey, and d_l and d_r are the distances to the left and right ears, respectively, then the volume of sound recorded in each ear is

$$s_l = \frac{S}{d_l^2} \quad s_r = \frac{S}{d_r^2}.$$

We don't know S , but some algebra yields the following approximation for the distance to the sound source

$$d \approx \frac{v(t_l - t_r)}{2} \cdot \frac{s_l - s_r}{2\sqrt{s_l s_r} - s_l - s_r}.$$

Thus, if we could somehow determine which sounds recorded in the left ear correspond to which sounds recorded in the right ear, and measure the difference in time in which each corresponding sound was recorded, we could estimate the orientation and distance to the prey emitting that sound. This is what our simulator ultimately will do.

To help you out, Louis has provided a procedure for computing d , given as inputs a difference in time between hearing the signal in each ear, the actual signal recorded in the left and right ear, the velocity of the signal, and the size of the baseline:

```
(define (compute-distance-and-angle time-shift lsig rsig v b)
  ;; time-shift is left time - right time, lsig and rsig are signal
  ;; strengths in two ears, v is signal velocity, b is baseline
  ;; procedure returns a list of distance and orientation of signal source
  (let ((theta (atan (- (* v time-shift)
                       (sqrt (- (square (* 2 b))
                                (square (* v time-shift)))))))
        (dist (* (/ (* v time-shift) 2)
                 (/ (- lsig rsig)
                    (- (* 2 (sqrt (* lsig rsig)))
                      (+ lsig rsig)))))
        (list dist theta)))
```

Louis has also provided default values, in arbitrary but consistent units, to use for the speed of sound and the owl's baseline: i.e. `signal-velocity` and `baseline` are global variables that you should use.

To model the owl's system, Louis decides to use infinite streams to represent the sounds recorded in each ear, where each element in the stream is a volume sample taken at regular intervals. Louis has also provided some procedures for generating test samples. Specifically, `make-test-signal` is a procedure of one argument, `noise`, that will return a pair, the `car` of which is the left ear's stream and the `cdr` of which is the right ear's stream. The argument `noise` allows us to specify how much auditory noise is present in the signal (good values to use in your simulations are in the range of 1–5). `make-test-signal` will generate a pair of streams in which a single sound is emitted by the simulated prey. Similarly, Louis has provided the procedure `make-test-signal-multi`, again a procedure of one argument `noise`, which will produce a pair of streams in which several sounds are emitted by the simulated prey.

Problem 1 Before we start building our owl simulator, it is probably useful to review some basic things about streams. Recall from the text that we could define an infinite stream of ones and integers using the following two expressions:

```
(define ones      (cons-stream 1 ones))
(define integers  (cons-stream 1 (add-streams ones integers)))
```

(a) Type them in and use the `print-stream` procedure to print them out. Use the `plot-stream` procedure to plot them on the graphics screen. (You needn't turn in anything for the `plot-stream` but show a transcript of your call to `print-stream`.) Notice that `plot-stream` does not first clear the graphics screen so you may need to do this yourself using `clear-graphics`. This makes `plot-stream` useful for plotting one stream against another, as we shall see later.

(b) Using the same idea (i.e., *without* using an auxiliary generating procedure), define the stream called `mystery-stream` that starts with 1 and computes each successive element of the stream as the sum of all the elements before it. Do you recognize the resulting stream? Warning: don't take advantage of the fact that you recognize it when you define it. Just construct it directly as prescribed above.

(c) Define a procedure which takes two integers as arguments, a lower bound `L` and an upper bound `U`, and returns a stream of random integers such that each number is greater than or equal to `L` but strictly less than `U`.

Hand in a listing of your definitions and a transcript of what the streams look like when you print them out using `print-stream`.

NOTE: For most of the remaining problems, you should consider solutions that take advantage of the provided higher order procedures, like `map`, `filter`, `add-streams`, and `scale-stream`. Also, your solutions should work on infinite streams. Do they also work on finite streams?

Problem 2 We're ready to start helping Louis with his owl simulator. As we suggested above, if we could find matching events in the streams for the left and right ear, we could use Louis' procedure to compute the range and orientation of the prey. But how do we find matching events?

Louis, as usual, is stumped, but fortunately Alyssa observes (as have others) that usually the onset and offset of a signal is quite noticeable compared to the background noise, and that one could simply look in the signal for a sudden change in volume. This, in fact, is simply the process of differentiation, which can be implemented by taking in a stream and producing a new stream in which each element is the difference between the successor of the corresponding element of the original stream and the corresponding element of the original stream. Thus, for example, if the first element is 5 and the second jumps to 42, then the first element of the `diff` stream should be 37.

Write a procedure called `diff` that implements this differentiation idea. What stream results when you apply `diff` to `integers`? to `mystery-stream`? As with every problem in this problem set, hand in a SCHEME transcript of your test cases.

Problem 3 Alyssa observes that the differentiation idea will accentuate sudden changes in volume. For example, differentiating the stream

1 1 1 1 12 12 12 ...

will lead to

0 0 0 11 0 0 ...

and in principle one could simply look for large values of the differentiated stream, such as the 11 above. In the presence of noise, however, almost every element in the differentiated signal will indicate some amount of change.

The trick is to separate the changes corresponding to the sound of the prey from the noise. To do this, Alyssa suggests using two ideas. First, she notes that one can often reduce the effects of noise in a signal by smoothing the signal. A simple way to do this is to average successive values in the stream, producing a new, smoother stream.

Write a procedure `smooth` that does this, and try applying it to your sample signals (namely, `integers` and `mystery-stream`). Try to define this using the higher order procedures like `scale-stream` and `add-streams`. Test it. You may find it useful to use `plot-stream` to plot both the original stream and the smoothed stream.

Problem 4 While averaging successive values does reduce the noise a bit, it may not be sufficient, so Alyssa suggests having the ability to smooth a signal repeatedly.

(a) Using `smooth`, write a procedure `smooth-n` of two arguments, a stream and an integer `n`, which recursively applies `smooth` `n` times. Test it on `integers`.

(b) Create a sample pair of signals using `make-test-signal` with some non-zero amount of noise and try first smoothing then differentiating one of the signals, then try differentiating then smoothing the same signal. Are the results the same

Problem 5 Using these two ideas, we could smooth each of the two signals, then differentiate them to get a stream of changes in the auditory channels. Now, we need to find places where there is a sudden sharp change in volume, which correspond to large positive or negative values in the differentiated stream.

The trick is to separate the real changes from the noise in the signal. Typically, we can consider the noise in the signal to be “white” noise (like the sound of an indoor waterfall). If we know the mean and deviation of that noise, than a traditional way of finding “significant” changes, x , is to find those changes whose absolute difference from the mean, m , is greater than some multiple of the deviation σ of the signal (typically $|x - m| > 3\sigma$).

Write a procedure called `mark`, which takes as arguments a stream (expected to be a smoothed, differentiated signal), a mean and a deviation, and which returns a new stream with a 1 in each place where there is a “significant” change, and with a 0 everywhere else. Applying `mark` to one of your smoothed, differentiated signals should result in a stream with a few places “marked”. (You can use Louis’ global variables `mean` and `dev` to try this.) Test this on a smoothed and differentiated stream. You may again find `plot-stream` useful.

Problem 6 Given that we can mark interesting points in the smoothed, differentiated signals, we now need to go back to the original smoothed signal and find the value of the signal at those points.

We want to create a procedure called `get-signal-strength` which takes two arguments: a marked stream and the smoothed (but undifferentiated) stream, with the following behavior. If the marked stream has a 1 at some point, the output stream should have the average of the value of the smooth stream at that point and the value of its successor. If the marked stream has a 0 at some point, the output stream should have a 0 at the corresponding point.

To do this, we are going to generalize the idea behind `add-streams`. In particular, write a procedure called `combine-streams`, which takes as arguments an operator and two streams, and returns a new stream, each element of which is obtained by applying the operator to the corresponding elements of the two input streams. Use this procedure to write `get-signal-strength`. As always, show a couple of interesting test cases. (For example, smooth `integers` (`n` times), then get the signal strength when marking with a deviation of 2. ...of 3. What do you expect the resulting stream to look like? Does it?)

Problem 7 Given the output of `get-signal-strength`, we need to record the significant events. We can do this by generating a new stream whose elements are a combination of the elements of the output of `get-signal-strength` and a label indicating the actual time sample in the stream (i.e. the first element is at time 1, the second at time 2, etc.), then generating a new stream in which we remove from this labeled stream all elements whose signal value is 0. The result is a stream of only the significant events, the entries of which are a combination of the point in the stream at which it occurred (i.e. the time) and the value of the signal at that point.

Using `combine-streams`, write a procedure called `dev-posns` which performs the described operation. (To get a stream of labels, we can use `integers`.) Test it. Notice that we cannot use `plot-stream` to plot the resulting streams since a `dev-posns` stream will not be a stream of integers.

Problem 8 Finally, we convert each element in this new stream to get the information needed to compute range and orientation. In particular, write a procedure `measure-time-spread-and-signals` which takes as arguments a left and right processed stream (as in the previous parts) and produces as output a new stream, each of whose elements is a combination of three things: (1) the difference in time between the corresponding element of the left and right streams, (2) the left signal value, and (3) the right signal value. Test this on the examples from problems 6 and 7.

Problem 9 We can put all of this together to build our first pass at an owl simulator. Write a procedure called `simulate` which takes as arguments a left and right stream, a smoothing factor (how many times to recursively smooth each signal – good values are 2–5), the signal velocity, and the baseline. It should do the following:

- smooth the two signals
- differentiate the results
- mark the significant events in those results
- get the signal strength in each smoothed stream associated with a significant event
- determine the positions of the significant deviations in the result
- combine the two results into a single stream, using `measure-time-spread-and-signals`
- generate a stream of range-orientation values by applying Louis' `compute-distance-and-angle` procedure to the result.

Try making a test signal using `make-test-signal` with moderate amounts of noise (e.g. 2 or 3) and then applying your procedure to the two resulting streams, using the global variables `signal-velocity` and `baseline` as arguments. At what orientation does the prey lie? Roughly, how far away is it?

Extra Experimentation/Credit For fun, you might try varying the free parameters in the simulator, namely the amount of noise you add to a signal, the amount of smoothing you apply to the result, and the size of the neighborhood you use for estimating the mean and variance of the signal. If you like, comment on any observations you can make about the effects of changes in these parameters on the performance of the system.