Homework No. 1 - Solution

EECS 290N
Fall 2004

Edward Lee
EECS Department
University of California at Berkeley
Berkeley, CA 94720, U.S.A.

October 5, 2004

A common scenario in embedded systems is that multiple sensors provide data at different rates, and the data must be combined to form a coherent view of the physical world. In general, this problem is called *sensor fusion*. The signal processing involved in forming a coherent view from noisy sensor data can be quite sophisticated, but in this exercise we will focus not on the signal processing, but rather on the concurrency and logical control flow. At a low level, sensors are connected to embedded processors by hardware that will typically trigger processor interrupts, and interrupt service routines will read the sensor data and store it in buffers in memory. The difficulties arise when the rates at which the data are provided are different (they may not even be related by a rational multiple, or may vary over time, or may even be highly irregular).

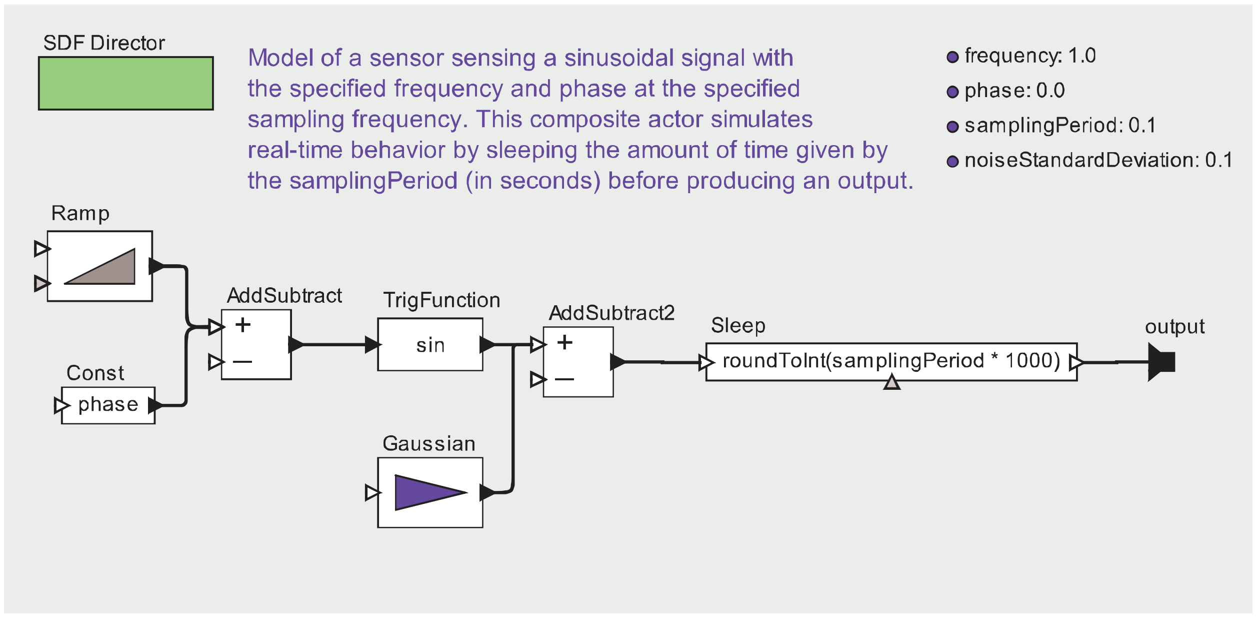
Assume we have two sensors, *sensorA* and *sensorB*, both making measurements of the same physical phenomenon that happens to be a sinusiodal function of time, as follows:

$$\forall\, t \in \textit{Reals}, \quad x(t) = sin(2\pi t/10)$$

Assuming time t is in seconds, this has a frequency of 0.1 Hertz. Assume further that the two sensors sample the signal with distinct sampling intervals to yield the following measurements:

$$\forall\, n \in \textit{Integers}, \quad x_A(n) = x(nT_A) = sin(2\pi n T_A/10)$$

A model of such a sensor for use with the PN director of Ptolemy II is shown below:


SDF Director
Model of a sensor sensing a sinusoidal signal with the specified frequency and phase at the specified sampling frequency. This composite actor simulates real-time behavior by sleeping the amount of time given by the samplingPeriod (in seconds) before producing an output.
frequency: 1.0
phase: 0.0
samplingPeriod: 0.1
noiseStandardDeviation: 0.1
Ramp
Const
phase
AddSubtract
TrigFunction
sin
AddSubtract2
Sleep
roundToInt(samplingPeriod * 1000)
output
Gaussian

You can create an instance of that sensor in Vergil by invoking the Graph → Instantiate Entity command in the menus, and filling in the boxes as follows:

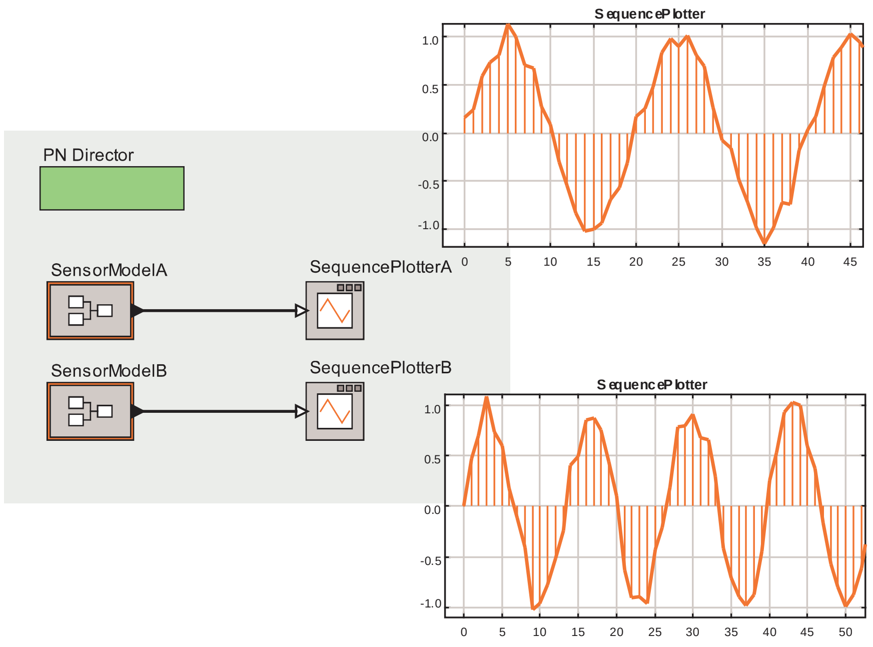
```
class: SensorModel
location (URL): http://embedded.eecs.berkeley.edu/concurrency/
                models/SensorModel.xml
```

Create two instances of the sensor in a Ptolemy II model with a PN director.

The sensor has some parameters. The *frequency* you should set to 0.1 to match the equations above. The *samplingPeriod* you should set to 0.5 seconds for one of the sensor instances, and 0.75 seconds for the other. You are to perform the following experiments.

1. Connect each sensor instance to its own instance of the SequencePlotter, found under Actors → Sinks → SequenceSinks in the library. Execute the model. You will likely want to change the parameters of the SequencePlotter so that *fillOnWrapup* is false, and you will want to set the X Range of the plot to, say, “0.0, 50.0” (do this by clicking on the second button from the right at the upper right of each plot). Describe what you see. Do the two sensors accurately reflect the sinusoidal signal? Why do they appear to have different frequencies?

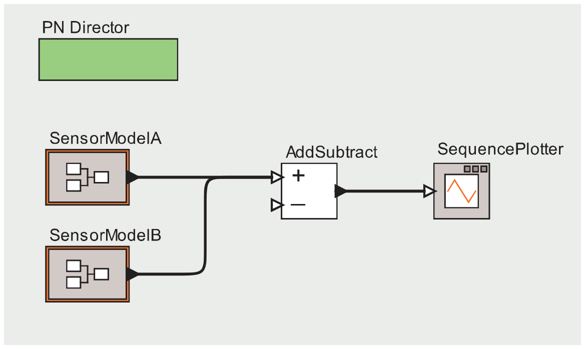
   Solution. The model and the resulting two plots are shown below:



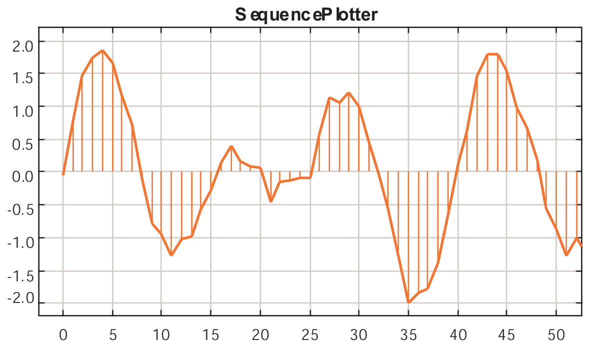
The plots show the (slightly noisy) sinusoids plotted vs. the sample number. Their frequencies (in cycles per sample) are, in fact, different, despite the fact that the SensorModel components purport to produce sinusoids with the same frequency. In cycles per second, the frequencies are roughly the same, although since PN has no time in its semantics, we have no assurance that the samples are generated at the rate we have indicated. □

2. A simple technique for sensor fusion is to simply average sensor data. Construct a model that averages the data from the two sensors by simply adding the samples together and multiplying by 0.5. Plot the resulting signal. Is this signal an improved measurement of the signal? Why or why not? Will this model be able to run forever with bounded memory? Why or why not?

   Solution. The model is shown below:



The plot resulting from a run is:

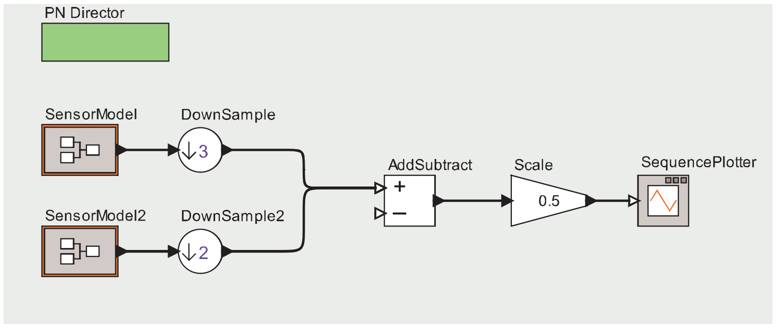


Notice that the plotted signal is not representative of the signal that the sensor models are sensing, which is a simple sinusoid. Instead, it is the sum of two sinusoids with different frequencies (plus noise). The AddSubtract actor is forcing the samples synchronize, adding one from each signal to get the result.

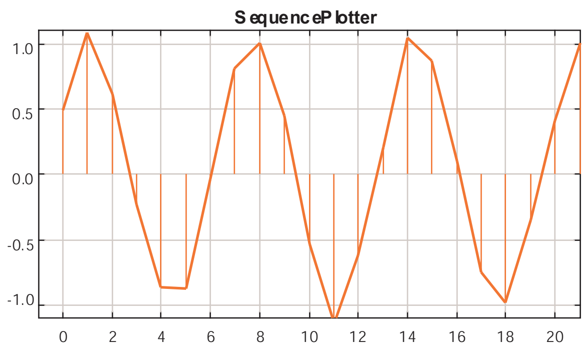
This model runs in bounded memory. In fact, the Parks strategy for maintaining bounded buffers results in the sensor models not working very well as sensor models. They use a Sleep actor to model the time between samples, but in PN, this only puts a lower bound on the time between samples. Because the two signals are added, the faster one will be slowed down to match the slower one by the blocking writes that implement Parks' strategy. □

3. The sensor fusion strategy of averaging the samples can be improved by normalizing the sample rates. For the sample periods given, 0.5 and 0.75, find a way to do this in PN. Comment about whether this technique would work effectively if the sample periods did not bear such a simple relationship to one another. For example, suppose that instead of 0.5 seconds, the period on the first sensor was 0.500001.

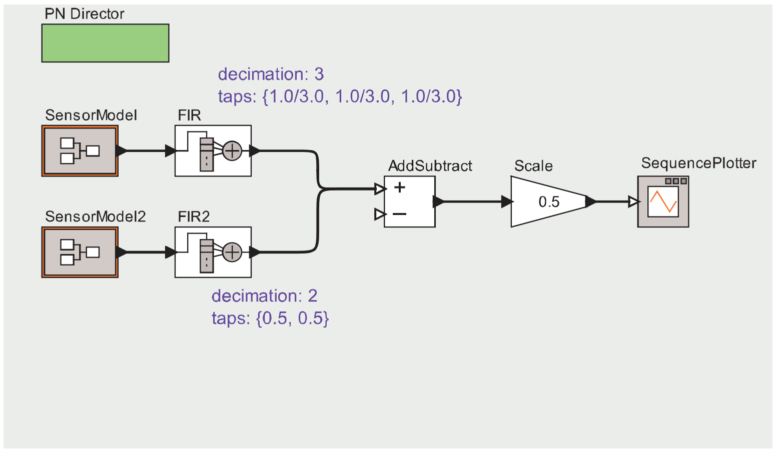
   Solution. One possible solution is shown below:



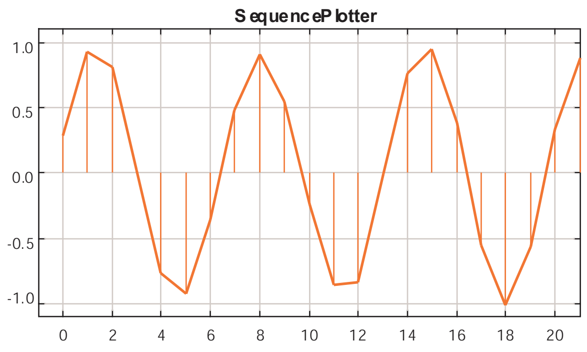
In this solution, we use our knowledge of the sample rates of the two sensor models to downsample the stream from each sensor to make their rates the same. Fortunately, the resulting downsampled sampling rate is still above the Nyquist rate for the sinusoid, so there will be no aliasing distortion for the sine wave. The result of running this model is shown below:



However, we are discarding useful information from the sensors by discarding samples. A more effective design uses FIR filters, as shown below:



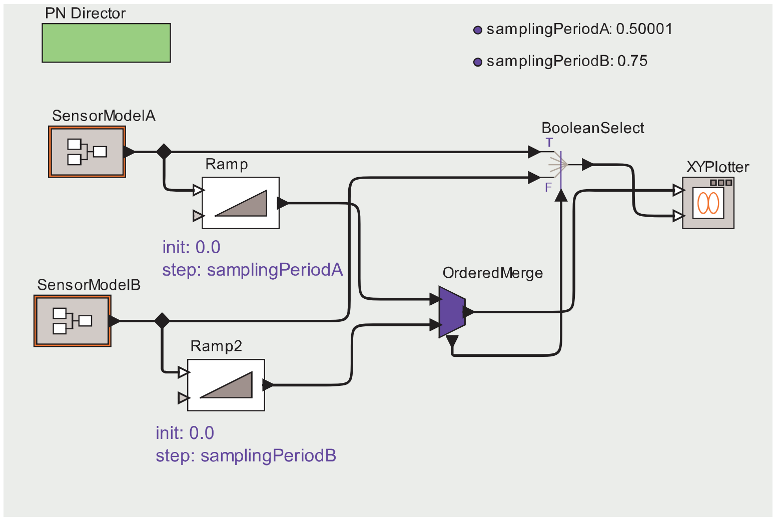
The non-default parameters of the FIR filters are shown in the figure. In this model, instead of discarding sensor data, each output of the downsampling filter is the average of either 3 or 2 neighboring samples. The result of running this model is shown below:



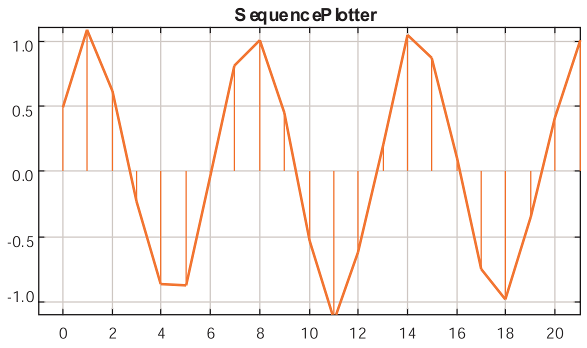
Neither of the two downsampling techniques above would work well if the sample rates were not such simple multiples. □

4. When sensor data is collected at different rates without a simple relationship, one technique that can prove useful is to create *time stamps* for the data and to use those time stamps to improve the measurements. Construct a model that does this, with the objective simply of creating a plot that combines the data from the two sensors in a sensible way.

   Solution. One possible solution is shown below:



In this solution, the two Ramp actors are used to generate a sequence of time stamps corresponding to the times of the original samples. The OrderedMerge will order these samples. Its output down the bottom tells us which input received each output that is produced. That signal can be used to merge the two sensor streams using a Select actor. The result of running this model is shown below:



This technique will work for any sample rates, as long as they are known *a priori*. However, this only goes as far as generating a reasonable plot. The samples are not actually combined in any useful way. In particular, even samples with the same time stamp are not combined, and instead appear in the plot as samples at the same time, but with different values. □