ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 9 — January 24, 2013

Patrick Lam *version 1*

Here are a bunch of simulations:

`http://www.youtube.com/watch?v=HUGjUvjtwS8`	physics N-body simulation
`http://www.youtube.com/watch?v=JGyJqXJWkuY`	aircraft
`http://www.youtube.com/watch?v=No5N6uYJaNk`	nuclear plant control room

Basic Idea. Create a model of a system to investigate its behaviour.

A *simulation* evaluates a mathematical model of a system to estimate the behaviour of the system.

These days, most simulations use computers to evaluate the state of the system. However, you could think of some board games as a primitive sort of simulation.

Why Simulate?

Building the system and testing on it always gives more accurate results. However, testing on the actual system is sometimes impossible:

- the system may not exist yet (e.g. proposed commercial jet design);
- the system may be costly to use (e.g. space shuttle);
- the system may be dangerous to use (e.g. nuclear reactor);
- the system may be disrupted by use (e.g. animal migration); or
- you just don't have enough access to the system (e.g. phones).

The next best thing, then, is to simulate. You might be forced to simulate when experimenting with an actual system is infeasible, too costly, too dangerous, or too disruptive to the system. Or you can simulate for fun and profit (eg a computer game).

Case Studies

We will discuss two case studies, both related to the robots that students programmed in previous offerings of this class. The case studies simulate the robots with varying degrees of accuracy.

Coarse Simulation. Here's one design for a Lego robot and environment simulator:

- Create a class `Board`. Other classes can query this class to find out whether the board is black or white at a certain point.
- Create a class `Robot`. This class stores, in fields, the current position and velocity of the robot, and contains the main logic of the robot.

- Create a class `LightSensor`. The `Robot` gets light readings from the `LightSensor` which are appropriate to its position on the `Board`.

Then you can have a main simulation driver which calls the `Robot` to (1) update its position according to the velocity; and (2) turn the robot if necessary. (You would probably program these actions in different methods.) Note that each call to the `Robot` would simulate the effect of time moving forward by one time-step.

Detailed Simulation: Visual Simulation Environment. It is possible to simulate the same system at many different levels of detail. For instance, Microsoft's Visual Simulation Environment simulates the behaviour of your robot code much more realistically.

In particular, VSE creates a simulated world and evaluates the behaviour of physical objects within that simulated world. This simulation resembles step (1) in the coarse-grained simulation, except that it tracks the behaviour of multiple objects and has a more-detailed model of the environment than just a `Board`.

VSE also calls your actual code to simulate the behaviour of your control software. This is like step (2) in the coarse-grained simulation, but in this case it is running actual code.

Caveat. Simulations aren't perfect. For instance, a previous lab report reported that everything worked fine in simulation under VSE, but needed lots more work to work on the actual robot.

Why?

The Android emulator is pretty good as far as it goes, but it's difficult for it to emulate accelerometer readings.

Other Examples

You might simulate a digital logic circuit using discrete techniques, since gates change values at specific times, in response to changing inputs.

On the other hand, you might simulate an analog circuit using continuous techniques, since the voltages and currents change continuously with respect to time.

In terms of writing a simulation, you would use different techniques to implement discrete simulations (queue-based: put the time of next event into the queue) and continuous simulations (numerically integrate an ordinary differential equation over and over; see ECE204).

Classifying Simulations. Here are three orthogonal axes along which you can classify simulations.

Continuous vs. Discrete Simulation Models:

- A *continuous* simulation model represents a system with continuously changing state variables.

- A *discrete* simulation model represents a system with discrete state changes.

Dynamic vs. Static Simulation Models:

- A *dynamic* simulation model represents a system as it evolves over time; it recomputes the state of the system at various times.
- A *static* simulation model is a one-shot deal: you feed it some conditions and it tells you the result. This is useful for what-if simulations: if I price my cookies at $2, how many will I sell?

Deterministic vs. Stochastic Simulation Models:

- A *deterministic* simulation model exactly computes the simulated state of the system at every step (subject to inaccuracies in the calculations).
- A *stochastic* simulation model uses randomness to (accurately!) estimate the expected behaviour of the system without computing the behaviour of each particle; it relies on the underlying randomness of the system components.

Simulation Tools. While you'd have to develop your own tools in certain domains, others domains have well-supported tools. These tools allow you to rapidly prototype a simulation of a complex system and its environment. However, you must understand how the tool works before relying on it. Here are some examples:

- *Microsoft Visual Simulation Environment* simulates decentralized software services for robots.
- *Arena* simulates businesses, services, and manufacturing processes.
- *Simulink* simulates a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing; people use it to investigate and verify implementations of these systems.
- *SPICE* and its many variants can be used to simulate analog circuits.

General-purpose simulation languages also exist to aid in the development of complex, application-specific simulation tools.