Chapter 5
Modeling Computational Systems

How do we model digital computers operating in a physical context? This chapter looks at the sense-compute-actuate model of implementing controllers. It addresses the physical aspects of implementing computation, both for analog computers and digital computers, and covers quantization (and quantization levels), discretization (and sampling), the Nyquist–Shannon Theorem, embedded hardware and software, and real-time systems and constraints.

5.1 Introduction

Now we want to implement control systems. How do we do that? We need to build a machine! Often, such a machine includes a sensor, an embedded computer, and an actuator. An embedded computer can be just a regular computer that has been turned into a dedicated machine inside another device, such as a watch or a car. Such systems often have demanding constraints in terms of size, energy utilization, unit cost, reliability, and real-time responsiveness. Meanwhile, real-time responsiveness typically means that the system must respond to some inputs (such as commands from a user) within a certain time limit.

We can, in principle, implement a controller with an analog circuit consisting of resistors, capacitors, and inductors. In fact, at one time people did just that. Nowadays, however, it is more common to use digital circuits. Why did this change happen? It is instructive to consider the general circumstances that created the right conditions for this fundamental shift in computing technology to take place. For example, analog computers can be quite fast and versatile. Why would they be phased out?

By and large, the biggest concern with analog computing was reproducibility. Performing a computation precisely required precise parameters to components. Not only that, but these components and their characteristics were sensitive to temperature and surrounding electromagnetic fields, which themselves can be a function of the ambient environment or how the components themselves are used and connected. To
make matters worse, over time, most the characteristics of most components changed as they aged.

Digital circuits use analog electronic components but, in contrast to analog circuits, their primary function is to connect them and build circuits out of them to transmit, process, and output only two levels of voltage. These two levels represent a binary digit. Any value outside those two levels is considered an indicator that the system has not yet finished transitioning between those two levels.

Our first impression of such a system might be that it seems like a vast underutilization of electronics: Instead of cherishing the continuum of possible values that an electronic circuit produces, and viewing it as a distinct value, we are concerning ourselves only with two levels (or, more precisely, two ranges of levels) that represent a single binary digit.

This design choice of working with “two levels” also means that we compute only with “quantized” values; that is, values that have been explicitly turned into a finitary, discrete quantities or representations. The big advantage of this is that it enables the transition from working with components that have uncertainties about their characteristics, to circuits where an answer is clearly either right or wrong. This has made it possible to understand and overcome the challenges of building very fast, small, and energy-efficient systems that are nevertheless deterministic and highly reliable. The broad success of this transition is witnessed by the pervasive presence of digital computing technologies around us today.

5.2 Quantization

Quantization helps make computing with circuits robust. In essence, quantization is the mapping of ranges from an infinite set to ones in a finite set, usually preserving order. Digital circuits are quantized in their voltage ranges. In binary digital circuits, there are just two main ranges, one below a certain level, and one above, with the middle being viewed as an invalid, transient state. Quantization is one of the key ideas that makes it possible to build highly reliable digital systems out of analog components that may, individually, be much less reliable.

Conceptually, the basic building blocks for digital computers are gates that operate on binary inputs to produce binary outputs, such as NOT, AND, and OR gates. The following table introduces some standard textual and graphical notation, as well as the relation that these gates impose between inputs and outputs:¹

---

¹ This table is derived from the article Logic Gate.
In actual electronic circuits such as those on an integrated chip, gates are realized (or “implemented”) using transistors, which are continuous systems. However, they are designed to be “stable” only for certain “high” and “low” ranges of values that correspond to a 0 or a 1 (not necessarily in that order). Then, entire circuits built out of these gates are designed to operate in a reliable manner only for such special values that correspond to 0s and 1s.

The exclusive use of only two ranges of values (one corresponding to 0, the other to 1) immediately brings up a fundamental problem: If we need infinite precision, then we need an infinite number of bits to represent the number. More often, builders of digital computers adopt a finite representation for values. This is a challenge that we must address whenever we use a digital computer.

### 5.3 Discretization: How Fast Can Your Circuit Go?

From the time that the inputs arrive, each gate needs a minimum amount of time before it can produce the correct value on the output wire. Such propagation delays can range from picoseconds to tens of nanoseconds, depending on the technology used to create them. No matter how minor such delays may be, any combination of gates will also have to take some time before the right answer is produced. When all propagation of signals in a circuit is completed, it is said to be in a **stable state**. Any well-designed circuit should have a clearly defined time after which we can guarantee that it has had enough time to stabilize.\(^2\) Most digital circuits have this well-defined

---

\(^2\) Note that not all circuits can become stable. For example, some circuits are used to create periodic signals (such as a clock signal) that are meant to oscillate indefinitely between 0 and 1.
value, but this introduces another limitation of digital computers. In particular, it determines the maximum rate at which the circuit can sample (and process) external inputs.

To support memory and iteration, digital computer circuits generally contain wiring “loops” that feed certain outputs back as inputs to the circuit. As a result, determining the amount of time that a circuit needs to produce the final, stable output can be challenging. Circuits that are created by freely combining gates are generally called asynchronous circuits. An alternative strategy is to use certain wires as clock wires to make groups of gates work in a lock-step manner. Such circuits are called synchronous circuits and clocking tends to simplify and help organize their design. Both circuit types have advantages and disadvantages for different applications, but predominantly synchronous circuits are used in microprocessor design today.

The typical approach to modern circuit design involves having a single clock that drives the operation of the whole circuit. The clock rate puts an absolute bound on the maximum possible sampling rate. There are, however, other considerations (such as available memory and the rate at which the information can be processed by the rest of the circuit) that may lower the maximum possible sampling rate. This means that any system implemented using a digital computer can only observe a continuous signal at specific times, and with a minimal gap between such samplings. The process of mapping continuous time to such a (countably infinite) notion of time is called discretization.

5.4 Detour: Boundedness of Digital Memory

Binary gates can be used in a variety of ways to construct memory cells by using the idea of feedback in a digital circuit. In particular, consider an AND gate with two inputs A and B, and an output C. Now consider the situation where we connect the output C back to the input B. The result is that we have a circuit with only one valid input A (the B input is already “plugged” with the output C). However, C can still be viewed as an output (as we are allowed to connect one output to several inputs). How would such a circuit work? To answer this question, we must consider the behavior of the system not only based on what the single input A is, but also on whatever the current value for the output C happens to be. A careful analysis would show that if we ever start with a situation where the input was 1 (or True), then it would stay that way as long as the input A was also 1. Once the input A changes to a 0, the output would be forced to switch to 0 and would stay that way forever, no matter what value we input for A.

This is a simple circuit that exhibits a very basic type of memory. In a way, it is a single-event memory: Assuming that we started it in a state where the output was 1, it would “remember” any occurrence of a 0 input by immediately turning the output to 0, and staying that way forever.

The exercises in the Study Problems section will show us, among other things, that storing one bit of memory takes a few gates and a bit of wiring. Furthermore,
as long as our CPUs, external memory, and external storage devices are finite, our
digital computers can only store a finite amount of information, although the cost of
such storage seems to be perpetually decreasing.

5.5 Detour: From Hardware to Software—Storing Executable
Commands in Memory

So far, this chapter has focused on hardware. The reason for this is that the most
fundamental effects that arise when we connect an embedded digital computer to
a physical system, namely, quantization and discretization, are due to the nature of
hardware rather than software. Of course, software can contribute to the extent to
which these effects occur, and can certainly help to mitigate them. Furthermore, these
effects have a significant influence on the way embedded software is built, giving
rise to the need for responsiveness, real-time deadline, reliability, fault tolerance, and
many other features. But the goal of this chapter is to help the reader develop a sense
for the nature of the problems that arise and for how these problems can manifest
themselves in the overall behavior of basic Cyber-Physical Systems. After mastering
the material discussed in this chapter, the To Probe Further Section will introduce
the reader to some basic concepts relating to the development of embedded systems.

5.6 The Effect of Quantization and Discretization on Stability

It is clear that quantization and discretization are two new issues that must be
addressed in the process of implementing a controller. Both control theory and
embedded systems techniques provide conceptual methods for developing such im-
plementations. For the most part, these topics are beyond the scope of the current
chapter. What we want to do instead is to help the reader develop an intuitive un-
derstanding of the effect of quantization and discretization on the operation of a
controller.

5.7 Abstract Modeling of Computational Effects

Interestingly, there is no need to switch completely to a specific hardware and
software platform, and to model in detail what happens when we implement the
entire controller using a digital computer. Rather, we can get a pretty good idea of
the effect of these two transformations by making small modifications to a model of
a continuous controller.

We will consider the effect of quantization and discretization on a controller
attached to a point mass. Without any quantization or discretization, the idealized
controller produces a force proportional to 10 times the distance of the mass from the original:

1. error = 0 − position,
2. controllerforce = 10 * error,
3. f = ma ⇒ a = f/m = controllerforce/1 .

In the first equation we are saying that we are taking the goal position as 0. Thus the error will simply be the negative of the position. Equations 1, 2, and 3 together imply that a = x'' = −position = −10x. Thus, the equation for the entire system (mass and controller) is x'' = −10x. This is an idealized, continuous system that also happens to be critically stable, as the system continues to oscillate indefinitely. In Acumen, this system can be modeled as follows:

```plaintext
model Main(simulator) =

Initially
x = 1, x' = 0, x'' = 0

Always
x'' = -10*x
```

Note that the equation above is a very compact representation of the system we described above (proportional control). We can introduce extra intermediate "dummy" variables to point to the places where an idealized "sensor" and an idealized "actuator" would be carrying a signal of a certain value. It is enough for our purposes here to identify a value representing what the sensor reads, and we can do it by modifying the Acumen model above as follows:

```plaintext
model Main (simulator) =

Initially
x = 1, x' = 0, x'' = 0,

Sensor = 0

Always
Sensor = x,

x'' = -10*Sensor
```

This model captures essentially the same behavior as the first model. It can be seen as representing an idealized sensor that can read the exact position of the mass. Naturally, in practice, converting a physical quantity (such as a process) into a signal representing this value on a wire is a non-trivial process. In fact, it is generally impossible to capture such a quantity exactly and without delay. So, both models should be viewed as representing a highly idealized system.
5.8 Modeling Quantization

Now we refine our model to reflect the reality that sensed values are generally represented in a quantized manner. We can model the quantization process as a continual process of trying to “track” the continuous value by means of a “sensor value” that can only be changed by fixed “quanta.” The following model captures this tracking process:

\[
\text{model Main (simulator) =}
\]

\[
\text{initially}
\]

\[
x = 1, \ x' = 0, \ x'' = 0,
\]

\[
sensor = 0
\]

\[
\text{always}
\]

\[
\text{if } ((\text{sensor}+0.03)<x) \text{ then sensor}^+ = \text{sensor}+0.3
\]

\[
\text{else if } ((\text{sensor}-0.3)>x) \text{ then sensor}^+ = \text{sensor} -0.3
\]

\[
\text{noelse,}
\]

\[
x'' = -10*\text{sensor}
\]

Here we have replaced the simple \(\text{sensor} = x\) relation with a more involved relation that moves the sensor values in steps of 0.3 in such a way that they are never more than 0.3 away from the actual value. This means that, while it is possible that there is an error in sensing, in terms of the difference between the actual value of \(x\) and our representation of the value of \(x\), this error is bounded. This is representative of what happens whenever we have to quantize a value. The result of this quantization propagates throughout the system. First, the value of the sensor only changes when the value of \(x\) has changed to be far enough from the current value of the sensor to trigger a change. Second, it is worth noting that this also means that the value of \(x''\) will only change when the value of the sensor changes. Thus, there is no real need for us to use a continuous assignment for the relation \(\text{sensor} = x\), and we can convert it into purely discrete assignments in the branches of the if-statements. This makes it easier to see that what we have now has more quantization than may be evident at first glance, but it is not strictly necessary. In fact, it is convenient to be able to mix discrete and continuous assignments in the manner above, in order to analyze the effect of quantization and/or discretization at very specific points in what would otherwise be a continuous-time system.
The signal for sensor and \( x'' \) will therefore have discontinuities. However, the resulting signal for \( x' \) and \( x \) will not have such discontinuities, as the integration relation that determines them based on the higher derivatives will smooth out such jumps.

The most significant effect of the quantization can only be seen by simulating the model and observing what happens to the variable \( x \): with the addition of quantization, the system is no longer stable. Intuitively, we can view quantization as having introduced a type of delay, whereby the system does not really see the value of the input until that input has sufficiently changed from the last reading. Of course, that change itself takes time, and that is what leads to the delay. Thus, the instability that we see here is very similar to that seen in systems where a delay is introduced. In many situations, we can overcome this instability by improving the operation of our controller to have a more stabilizing effect on the resulting system. For linear systems, we can more precisely quantify the effects of discretization and incorporate them accurately in the design of the entire system. For non-linear systems, more specialized analysis is needed for different kinds of systems.

To quantify the “amount of instability” in this example, we can simply look at the maximum height of the last wave at the end of the simulation. In this case, the last wave has a height of last full peak of \( x \) that we see in the simulation. Its height is about 3.4. This can be viewed as a “gain” of 3.4 times in the system’s oscillation, because the original system (without quantization) had a signal with a maximum height of only 1.0.

### 5.9 Modeling Discretization

We now turn to discretization. Discrete sampling of continuous signals can be seen as a way of taking “still photographs” of a moving object. The key to being able to take such samples is to have a mechanism for triggering the recording of such values. This requires a variable (such as the sensor in our previous example) to record the value, and an event to trigger the writing of the value of the external continuous variable into the computational units representation of that variable. Sampling (and discretization) can be periodic (occurring with equal gaps between samples) or, more generally, event-driven. Periodic sampling can be viewed as event-driven sampling that is triggered by a clock event that occurs periodically. All we need to model this type of sampling is a “bucket” that increases at a fixed rate, and the ability to trigger the sampling when the bucket reaches a certain threshold. The following model represents such a situation, where the threshold (sampling period) is 0.05:
model Main (simulator) =

initially
x = 1, x' = 0, x'' = 0,
sensor = 0,
bucket = 0, bucket' = 1
always
bucket' = 1,
if (bucket > 0.05)
then bucket+ = 0, sensor+ = x
noelse,
x'' = -10*sensor

Just as with the previous example, the variable sensor now changes with discontinuities, and so does $x''$. The difference now is that the changes occur at a fixed frequency.

To quantify the amount of “instability” introduced by discretization, we can take note of the height of the last full peak in the simulation, which is 2.6. Again, we can view this as a “gain” in the amount of oscillation in the system, because the original system (without discretization or quantization) had a maximum height of 1.0.

5.10 Detour: Discretization, Sampling Rates, and Loss of Information

It can be difficult to capture all the information in a dense-time signal by a finite number of samples (which we can think of as a discrete-time signal). To convince yourself of this fact, consider any discrete-time signal, and then construct two different dense-time signals that pass through all the same points but are slightly different between any two points of your choosing. Note that it is useful that this difficulty only exists when we consider all possible functions; if we are willing to restrict ourselves to certain classes of functions, the situation improves dramatically. For example, we often only need to consider signals that have a maximum frequency component. This restriction can be viewed intuitively as saying that the signal “does not change faster than a certain rate.” More technically, it means that the frequency-domain representation of the signal is zero beyond a certain frequency. In many cases this is a very reasonable assumption, because many physical systems can be
viewed as “low-pass” filters that essentially ensure that this requirement is true. For such systems, the Nyquist–Shannon sampling theorem has good news for us: If the maximum frequency component in a signal is $B$, sampling that signal at $2 \times B$ is enough to capture all the information in that signal.

Two remarks are necessary in relation to this important and widely cited theorem. The first is that, in the context of building control systems, it is only telling us that no information is lost in the sampling. It is a different matter to ensure that the computation performed on this signal does not introduce additional loss. The fact that no information is lost in the signal does not mean that performing a naive analog of the dense-time computation is the right thing to do to get corresponding behavior. The second remark on the theorem is that it is certainly not the only situation where a system can be reconstructed from bounded-rate samples. There are many instances where, based on different kinds of assumptions about a signal, limited sampling can lead to significant or complete information about the signal.

5.11 The Effects of Quantization and Discretization Easily Compound

So far we have considered quantization and discretization separately, but in a setting where we use a digital computer to implement a controller, we have both effects. It is easy to model the effect of doing both things at the same time as follows:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x' ' = 0,
sensor = 0,
bucket = 0, bucket' = 1

always
bucket' = 1,

if (bucket > 0.05)
then if ((sensor + 0.3) < x)
then sensor+ = sensor + 0.3
else if ((sensor - 0.3) > x)
then sensor+ = sensor - 0.3
else bucket+ = 0
```
no else,

\[ x'' = -10 \cdot \text{sensor} \]

It is highly instructive to note that the gain now becomes 7.8, which is noticeably larger than it was for just the quantization or discretization effects taken separately. This example reminds us of the importance of taking into account both quantization and discretization in the design of a system. It also illustrates the importance of ensuring that actual implementations of a control system on a digital computer are able to realize the sampling rate (or rates) upon which the design of a particular Cyber-Physical System is based.

5.12 Chapter Highlights

1. Modeling Computational Systems
   (a) We started from a continuous substrate
   (b)Introduced hybrid systems
   (c) Most computational systems today are not “continuous”
      • Discretized
      • Quantized

2. Quantization
   (a) To represent at zero’s and one’s
      • How computers work
        – Basic gates
        – These components are really analog!
   (b) Why do we work with zeros and ones?
      • More reliable than analog computers
      • Components can be much cheaper, much smaller
      • Components can be composed more easily

3. Discretization
   (a) To work at discrete time steps
   (b) Why do we work at discrete time steps?
      • Timing is easier to analyze
        – Avoid race conditions
        – Avoid complexities with feedback

4. Effect on stability of control systems
   (a) Discretization
   (b) Quantization
(c) How can we manage this?
- Pick resolution
- Pick sampling rate
- Tune gains

5.13 Study Problems

1. Extend the circuit described in Section 5.4 with an extra input D that causes the output C to become 1 when that input has the value of 1. You may choose up to two additional gates to use.

2. Extend the circuit described in Section 5.4 with an extra input CLK that causes the output C to take whatever value the input has when the input CLK is 1. You can assume that the input A does not change during the time that the input CLK has the value of 1. When the input CLK is 0, the output stays the same independently of the behavior of input A.

3. Run the model of Section 5.7 in Acumen and confirm that it produces the oscillatory behavior described above. Determine the period of the signal based on the Acumen simulation. Assuming that the signal is a cosine wave, confirm your determination about the period by substituting it into the equation and checking the result.

4. There is one difference between the sensor signal and the $x$ signal in the last model of Section 5.7. Can you spot it?

5. In Section 5.8 there is a claim that “the resulting signal for $x'$ and $x$ will not have such discontinuities, as the integration relation that determines them based on the higher derivatives will smooth out such jumps.” Confirm these observations by running the model appearing before this claim in Acumen. Modify the code to determine the shortest time between two different changes of value to the variable sensor. Hint: It is OK to introduce your own timer into the model to compute this. Also, you only need to compute the shortest time for the transitions that actually occur within the duration of the simulation.

6. Modify the last model in Section 5.9 to determine the size of the biggest jump in the value of the sensor during the simulation.

7. Modify the controller in Section 5.11 to damp the system by taking into account the speed of the mass in addition to its position. You can use the variable $x'$ on the right-hand side of the equation for $x''$ initially. However, keep in mind that $x'$ cannot “magically” appear inside a digital computer. To address this problem, your final model should include a method for computing an estimate of speed based only on the value of the variable sensor.
5.14 Lab: Stability Exercises

The purpose of this lab is to review and expand on the discussions in this chapter on quantization and discretization with an emphasis on connecting it to what we have learned so far about dynamical systems and control. Working through the activities of the lab will prepare us for the issues that need to be addressed by the upcoming project activities, and more importantly, with how to have a basic appreciation of the effects of quantization and discretization on cyber-physical systems.

In this chapter we have already seen the following model:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x'' = -10,

always
x'' = -10 * x
```

This is a valuable model for us because it can be viewed as a classic example of a control system with negative feedback, and where the resulting behavior is critically stable, that is, just in between being stable and unstable.

Simulate this model to confirm that the result is that $x$ is oscillating around zero.

We also noted that the following variant is essentially the same, but with a variable sensor introduced to indicate what we view as the sensor input to the controller expressed by the equation:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x'' = 0,

sensor = 0

always

sensor = x,

x'' = -10 * sensor
```

This variant is a good starting point to explore how to represent what happens to a signal as it is measured and passed on to the process that computes the control signal. Note that by simply setting the sensor signal to the quantity we want to measure we are making an assumption that the sensor is highly idealized. It will take additional modeling and transformation of this mapping to have a more accurate representation of what happens in reality when a signal is measured.
Quantization results from the fact that we are using machines that use bits as discrete representations of values. We need not worry about the details of how such representations are realized to model quantization in a simple fashion, and to explore its effect on a basic control system such as the one we are considering here. The following model illustrates a basic method for modeling quantization:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x'' = 0,

sensor = 0

always

if (sensor + 0.3) < x
then sensor+ = sensor + 0.3
elseif (sensor - 0.3) > x
then sensor+ = sensor - 0.3
noelse,

x'' = -10 * sensor
```

Notice that we have removed the direct couple between `sensor` and `x` which was provided by one equation in the previous model, and replace it by an `if` statement. This statement compares the value of `sensor` to the value of `x` and corrects any large difference by incrementing or decrementing `sensor` by discrete steps to get it as close as possible to the value of `x`. This is not typically how sensors work, but it illustrates the ease with which quantization can be modeled.

Before simulating this model, write down a description of what the `sensor` will look like. Also, write down a description of the impact that you expect this quantization will have on the overall system behavior. Then run the simulation. Were your expectations accurate? Were there aspects of the simulation plot that you did not expect or did not write down ahead of time?

If we consider the plots resulting from this simulation, we will notice that in 10 s (that is, by the end of the default simulation time in Acumen) the amplitude of the oscillation has grown significantly. Thus, this example illustrates that quantization can have a destabilizing effect on a control system. Before investigating any techniques for mitigating this effect, it is useful to make a mental note of the fact that this is only an example. It is quite conceivable that quantization could also have a stabilizing effect on systems.

Now let us go back to our template model (the one where we first introduced the variable `sensor`) and consider how we can model quantization. There are several ways in which this can be done, including updating sampled values based on changes
in the values being sampled, which can lead to an event-based model. Here we will illustrate quantization based on a clock. The following model gives an example of how this can be achieved:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x'' = 0,

sensor = 0,
bucket = 0, bucket' = 1

always

if bucket > 0.005
then bucket += 0,
    sensor += x

noelse,

bucket' = 1,

bucket'' = 1,

x'' = -10 * sensor
```

Here we have the variable `bucket` acting as a quantity that gets filled at a constant rate. When the value of `bucket` passes 0.005, it is reset to zero (thus emptied) and the value of `x` is sampled by copying it into `sensor`.

Notice that here we are quantizing only the sensing and not necessarily the calculation of the control equation of the update of the control output. For simplicity, they are modeled by a simple equation that holds all the time. However, since it is a simple linear calculation that depends solely on the value of `sensor`, it also gets revised whenever `sensor` is revised.

Before simulating the model, write down what you expect to see in the signals for `bucket` and `sensor`, as well as a description of what you expect will be the impact of discretization on the overall system. Simulate the model above, and then make a note of whether all your expectations were met, and whether there were any unexpected features in the result of the simulation.

If we consider the resulting plots from the last model we will notice that by the end of the first 10 s the amplitude of the oscillation goes up significantly, which is roughly the same as what we saw with quantization. The closeness in the magnitude of both gains is a coincidence. The key point is that for this system, both quantization and discretization have a destabilizing effect.
Naturally, we may be interested in modeling both effects at the same time, which is achieved by the following model:

```plaintext
model Main (simulator) =

initially
x = 1, x' = 0, x'' = 0,
sensor = 0,
bucket = 0, bucket' = 1

always

if (bucket > 0.005)
then if ((sensor+0.3)<x)
then sensor+ = sensor+0.3

elseif ((sensor-0.3)>x)
then sensor+ = sensor - 0.3

else bucket = 0

noelse,

bucket' = 1,

x'' = -10 * sensor
```

As may be expected, combining both effects also leads to an increase in destabilization of the system.

A natural way to reduce these effects is to increase precision and sampling frequency. In general, however, these can be directly proportional to the cost of the computational resources needed to perform the computation.

An alternative approach that can have lower costs is to introduce damping into the system. If this can be done, it can stabilize the system, albeit possibly at the cost of accumulating more delays into the overall system behavior.

To introduce damping, however, there is another important challenge: In the examples we have seen before, this requires the use of speed. But we are in a situation where we are modeling sensing and have no direct access to the quantity itself, not to mention the rate at which it changes.

To estimate speed, extend the model with two variables, sensor_last and xp. Extend the model so that sensor_last always keeps the last update to sensor. Then, using the two sensor readings and the time-step for sampling, make an estimate of the speed. Use this estimate to update the controller to introduce a damping effect. Find an appropriate gain constant so that the system returns to being (approximately) critically stable.
Compare your estimate of speed to the actual speed variable $x'$. Are there consistent differences between your estimate and the actual speed? Can this lead to problems for the system? Are there well-motivated ways in which you can improve your speed estimation method?

### 5.15 Project: Quantization and Discretization

The goal of this project’s activity is to help you understand the use of estimators to improve the quality of data from the sensor. Your task is to take the player developed from last week and observe how its behavior changes when you switch to a new model that involves discretization and quantization effects for both actuation and sensing signals. With this model, you should find that the controller that you had previously developed under the assumption of more idealized sensing does not quite work as well any more. You want to develop an improved player that is able to handle this increased level of realism in the model. To address this, the new variable `estimate_ballv` can be used to overcome the discretization and quantization effect of the velocity of the ball. You will also need to estimate the ball’s velocity in a discrete manner.

In the context of this project’s activity it will be useful to reflect on how you have overcome this new challenge, paying particular attention to the design of your controller and the various state estimation methods being used by the player.

As usual, feel free to continue to develop all the capabilities of your player in creative ways. Neglecting any one aspect (such as the effects of quantization and discretization) would put you at a disadvantage, but to really excel you need to take a holistic approach to the development of your player, and to make sure that you learn as much as possible from past experiences.

Incidentally, the approach that we have encouraged you take in developing this project is to start with a high-level model that allows you to focus on some big picture questions and then gradually go to lower level models that allow you to address the lower level challenges as well. Reflect on whether this has been a helpful strategy for you, and whether other approaches could have worked better, or at least equally well. In particular, would it have been helpful to expose all of the problems that you need to address from the outset?

### 5.16 To Probe Further

- General introduction on real-time systems and embedded systems.
- Article on Nyquist–Shannon Sampling Theorem.
- Washington Post article before software patents are revisited by US Supreme Court.
- Watch Grace Hopper’s elegant explanation of what is a nanosecond.
• Find out about the conference named after Grace Hopper, and think about what you may be able to contribute there.
• Watch the video recording of Edward A. Lee’s lecture on *Heterogeneous Actor Models* in the Halmstad Colloquium series. You may also want to checkout *Synchronous Data Flow* after watching this lecture.
• Article on the *Philosophy of Computer Science*. 

**Open Access**  This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License ([http://creativecommons.org/licenses/by/4.0/](http://creativecommons.org/licenses/by/4.0/)), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.