Chapter 2
Signals

Lasciate ogni speranza, voi ch'entrate.
Dante Alighieri, The Divine Comedy

We all send and receive signals. A letter or a phone call, a raised hand, a hunger cry—signals are our information carriers. Pervasive computing systems also send signals (to wheels, loudspeakers or displays), and sense signals (like speech, images, heart activity, or temperature). These signals are by nature *continuous*, meaning that their level is known always and everywhere, and they are *analog*, meaning that this level is an infinite-precision real number. The problem is that these properties are in conflict with the discrete and digital character of computers. The price the real-world signals have to pay to be admitted and processed by a computer is to undergo a process called *digitization*. Digitization happens in two stages. First, *sampling* periodically measures the signal's level and produces a finite-length sequence of *samples* of infinite precision. Next, *quantization* converts these samples into a sequence of finite-precision numbers. Digitization is the task of an electronic device, called *analog-to-digital converter* (ADC). The reverse process, needed by a computer system to decipher the carried message and send signals back to the real world, is called *reconstruction*. Reconstruction happens in another electronic device, called *digital-to-analog converter* (DAC). High-quality digitization and reconstruction are needed, in order to preserve the information the signal is carrying. This chapter is an introduction to signals and their representation in pervasive computing systems, with a focus on audio and video signals.

2.1 Signals in the Wild

2.1.1 One-Dimensional Continuous Time Signals

Signals are variations of a physical quantity that encode information. In some pervasive computing systems, these variations happen in *time*. A speech signal, for example, initially arises in the human vocal tract as a variation of air pressure. The

N. Silvis-Cividjian, *Pervasive Computing*, Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51655-4_2

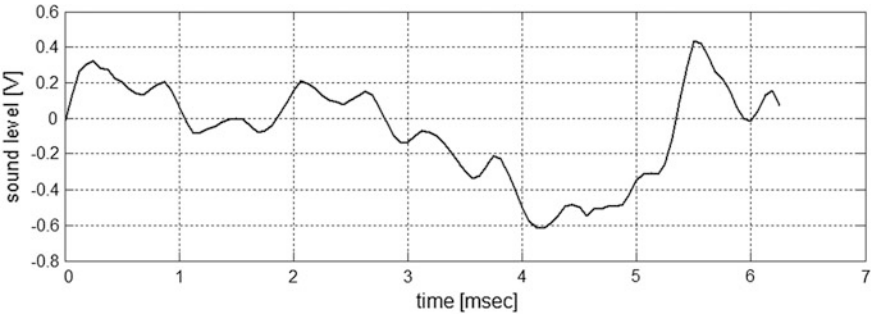


Fig. 2.1 Plot of a speech signal. The *vertical axis* represents the sound level, and the *horizontal axis* represents time

variation pattern propagates through air, until it eventually gets captured by a microphone, which transforms it in an electrical signal. Its level measured in volts (V), is proportional to the sound level, and can be plotted in a time waveform, as shown in Fig. 2.1.

The speech signal in Fig. 2.1 is an example of a *one-dimensional, continuous,* and *analog time* signal. Let us explain this long list of attributes. First, it is obviously a signal whose level varies in *time*. Second, it is *one dimensional,* because it can be represented mathematically as a function of a single independent variable, in this case the time. This mathematical function, denoted by *s,* assigns to each instant of time (denoted by *t*), a value, *s(t)*, equal to the signal level at that moment. Next, the signal is *continuous,* because this function is defined for *all* possible values of the variable *t, t* $\in \mathbb{R}$. Finally, because the signal level *s(t)* can be any real number in a continuous range, $s(t) \in \mathbb{R}$, the signal is called *analog*.

Typical one-dimensional time signals encountered in pervasive computing are: speech, acceleration, pressure, temperature, and biomedical signals used for electroencephalograms (EEG), electrocardiograms (ECG), or electromyograms (EMG). The patterns of variation in these signals can take different shapes. For example, the signal shown in Fig. 2.1 is a short fragment of a "Hello" recording. It looks like one of many—its level varying quite chaotically in time. Luckily, other sounds exist in nature, displaying a more interesting and organized behavior. One such example is the tone emitted by the tuning fork, a device used to calibrate musical instruments (see Fig. 2.2a to the left). In the same figure to the right, you can see a time plot of this sound. This plot is telling a story. To start with, one can clearly identify the moment when the tone was initiated as effect of the fork struck. Further, one can follow the sound evolution during a few seconds, until it eventually fades out. Notice that during these few seconds, the sound level has been gradually decreasing. This phenomenon is called attenuation.

The story goes on, if we zoom in this plot. By looking at Fig. 2.2b, we can discover for example that the signal has a specific pattern that repeats over equal time intervals. A signal with such an organized behavior is called *periodic*. Its

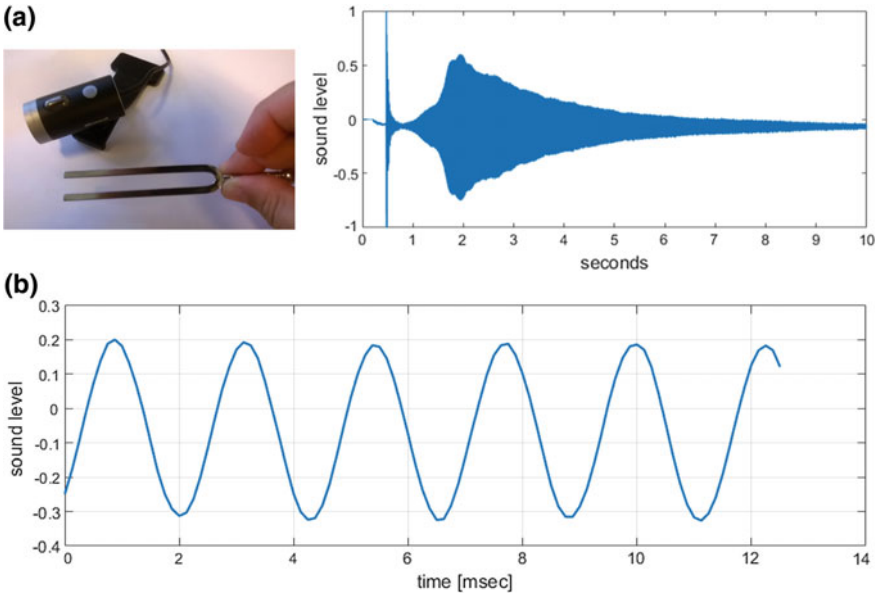


Fig. 2.2 **a** A 440 tuning fork experiment and a plot of the generated sound wave. **b** A zoom in the plot of the tuning fork signal

period, denoted by T, is defined as the length of this time interval. Mathematically, a periodic signal satisfies the condition $s(t + T) = s(t)$, for all $t \in \mathbb{R}$.

The *frequency* of a periodic signal, measured in Hertz [Hz], is defined as the number of identical patterns (cycles) contained in one second and can be calculated as:

$$f = 1/T\,[Hz] \tag{2.1}$$

where T is the period measured in seconds *[s]*. For example, a signal with the frequency $f = 10$ Hz will perform ten identical cycles each second.

Let us calculate the frequency of the tuning fork sound, using the formula (2.1). We measure on the plot that its period is approximately 2.3 ms, which corresponds to a frequency $f = 1/0.0023 = 435$ Hz. This is very close to 440 Hz, the frequency of the A note, as expected by any musician. Lucky us! We have just got a confirmation that the simple formula (2.1) does not lie; in contrary, it will save us in many situations, so try not to forget it.

The particular shape of the signal in Fig. 2.2b, which might look familiar from high-school math, is called *sinusoid*. *Sinusoidal signals* are one of the basic families of periodic signals, very useful in modeling common processes in nature. Therefore, they will deserve a special place in our story. Their name derives from *sinus*, or sine, a trigonometric function of a variable angle θ, plotted in Fig. 2.3. A few of its values for typical angles θ, expressed in radians, are:

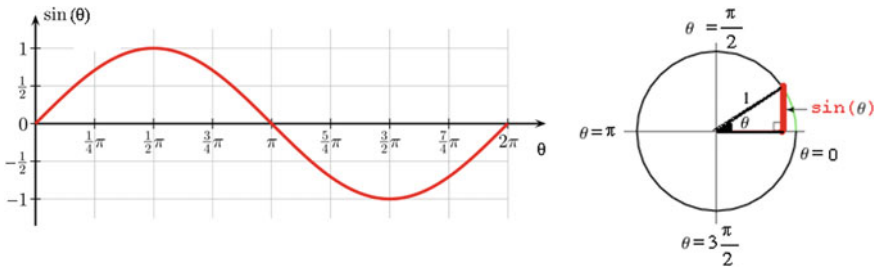


Fig. 2.3 Illustration of the sinus function

$$\begin{aligned} \sin(0) &= 0 \\ \sin(\pi/2) &= 1 \\ \sin(\pi) &= 0 \\ \sin(3\pi/2) &= -1 \\ \sin(2\pi) &= 0 \end{aligned} \tag{2.2}$$

This angle θ is in its turn, a function of time. A sinusoidal signal can be represented with the following mathematical formula:

$$s(t) = A\ \sin(\omega t + \varphi) = A\ \sin(2\pi f t + \varphi) \tag{2.3}$$

where A is called the *amplitude*, ω (pronounced omega) is the *radian frequency*, t is the *time*, f is the *signal frequency*, and φ (pronounced *phi*) is the *phase shift* that essentially depends on what moment in time we choose to call zero. All these parameters in this formula, except the time, are constant.

For example, Fig. 2.4 is a time plot of a sinusoidal signal, with an amplitude of 5 V and a period of $T = 2$. Its level oscillates between 5 and −5 V. The signal repeats the same pattern of oscillations every two seconds. Therefore, the frequency

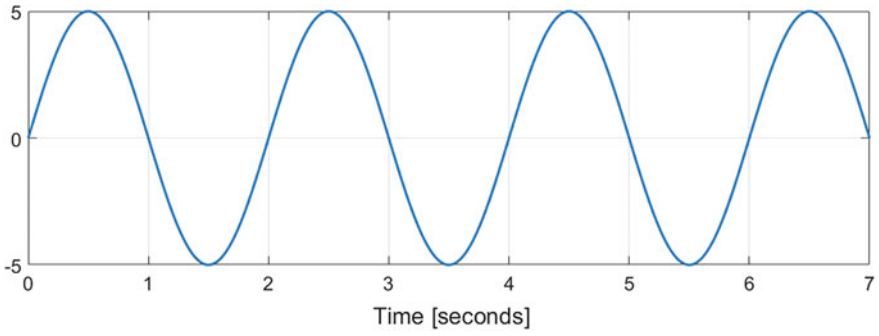


Fig. 2.4 A plot of a sinusoidal signal. Its parameters are: $T = 2$ s; $A = 5$; $\varphi = 0$; $S_0 = 0$

of this signal is $f = 1/T = 0.5$ Hz. The phase shift of this signal is zero ($\varphi = 0$), because the signal begins at time $t = 0$ with the value $s(0) = 0$.

Because at the initial moment $t = 0$, the signal level is not per definition zero. Therefore, the following formula describes a sinusoidal signal in a more general way.

$$s(t) = s_0 + A\ \sin(\omega t + \varphi) = s_0 + A\ \sin(2\pi f t + \varphi) \tag{2.4}$$

where s_0 is a constant component, called the *DC component* or *mean value* of the signal. The signal in Fig. 2.4 has a DC component equal to zero, meaning that the signal level oscillates between 5 and −5 V around the X axis, with a mean value equal to zero. If the DC component is for example 2 V, then the signal oscillates between 7 and −3 V.

2.1.2 Two-Dimensional Continuous Signals

We know by now how to mathematically represent continuous, time-varying signals. However, not all real-world signals are patterns evolving in time. Take for example a monochrome picture, also known as a black-and-white photo. Not the one that recently came out of a digital printer, but the one that dates back to the last century, developed in a darkroom, using traditional photography techniques, like the one in Fig. 2.5. This picture is a *spatial* pattern, represented mathematically by a *two-dimensional (2D) function*, denoted by $p(x, y)$. This function has two independent spatial variables, x and y, given by the position of each dot on the photo. The function assigns to each pair (x, y), the *brightness* at that position. Because this brightness value is known for each (x, y) in the continuous space on the photo, we say that the image signal p is *continuous*. The signal is also *analog*, because the brightness is represented by an infinite-precision real number, situated in the continuous range between 0.0 (black) and 1.0 (white). So now you can understand that

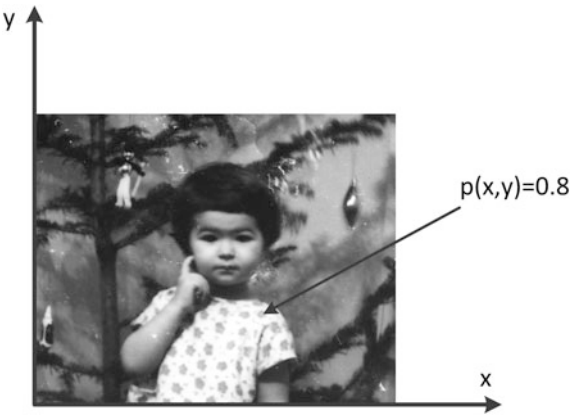


Fig. 2.5 An old black-and-white photo is a two-dimensional continuous, analog image signal

the photo in Fig. 2.5 is not exclusively black and white as its name would suggest, but can have any gray shade in between. Although this book will treat primarily 2D images, you should know that higher dimension image signals exist as well. Think of video camera signals, represented with a three-dimensional (3D) function of two spatial coordinates and time, or a tomographic movie of a beating heart, which is a 4D function of three spatial coordinates plus time.

2.2 Signals in the Cage

2.2.1 One-Dimensional Digital Signals

A time signal knocking at the computer's doors is blessed with two beautiful properties. First, the signal is continuous, meaning that its level is known at any time. Unfortunately, the computer is not particularly interested in this feature, because it does not have time nor space to record the signal at *every single moment*. In other words, "Big Brother" cannot watch you at all times. The only thing the computer can do is to poll the input signal from time to time. This process of measuring a continuous signal at isolated, equally spaced moments in time is known as *sampling*. The result is a sequence of *samples*, a *discrete-time* representation of the continuous input signal. Second, the input signal is *analog*, meaning that these samples are real numbers having an infinite precision. However, a computer, which is by nature digital, is unable to preserve this property, because it has only a limited number of bits available for data representation. In other words, "Big Brother" can by no means *precisely* know where you are. Instead, the samples are truncated to a finite-length digital representation. This process is called *quantization*. Sampling and quantization together are known under the name of *digitization*. This process describes the first thing that happens to the "wild" signal as soon as it gets captured by a pervasive computing system.

Digitization is a process that converts a continuous and analog time signal, s, into a *digital* signal, described by a discrete-variable, discrete-level function:

$$s[n] = s(n \cdot T_s) \tag{2.5}$$

where n is an integer between 0 and $N_s - 1$ representing the sample index, N_s is the total number of samples, and T_s is the *sampling period* defined as the distance in time between two successive sampling moments.

Formula (2.5) says in fact that the digital signal is a sequence (or vector) of finite-precision numbers, organized as follows. The first sample s[0] is the digital value of the analog signal at moment $t = 0$, $s(0)$; the second sample s[1] is the digital value of the signal at moment $t = T_s$, $s(T_s)$; the third sample is s[2], equal to $s(2T_s)$, and so on. Note that we use parentheses () to enclose the independent variable of a continuous–variable function, and square brackets [] to enclose the variable of a discrete-variable function.

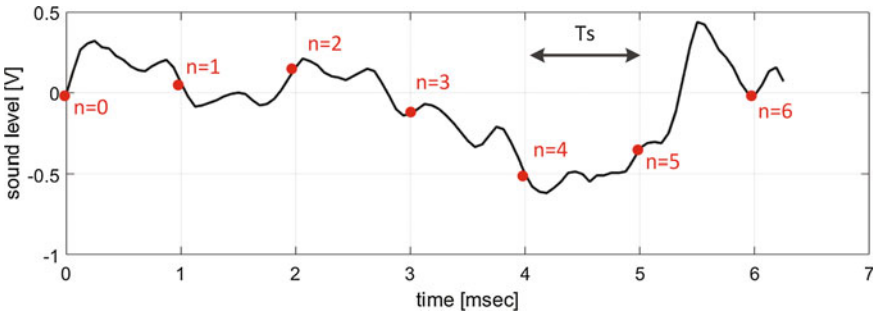


Fig. 2.6 Sampling the speech signal from Fig. 2.1. The *red bullets* represent the samples

Figure 2.6 illustrates the sampling process of the speech signal from Fig. 2.1. The samples, marked with red bullets, are taken every millisecond, meaning that the sampling period in this example is $T_s = 1$ ms = 0.001 s.

The *sampling frequency* or *rate*, defined as the number of samples taken every second, can be calculated as:

$$f_s = 1/T_s\,[Hz] \tag{2.6}$$

where T_s is the sampling period measured in seconds *[s]*. For example, the sampling frequency used to sample the signal in Fig. 2.6 is $f_s = 1/T_s = 1/0.001 = 1000$ Hz = 1 kHz.

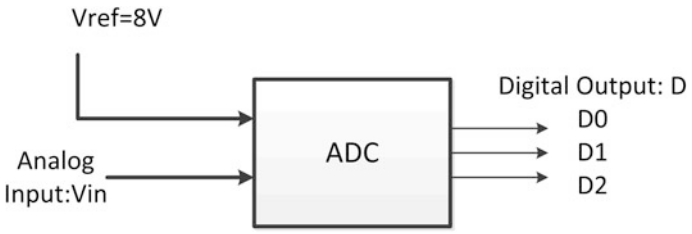
Digitization of a one-dimensional analog signal is the task of an electronic device, called *analog-to-digital converter (ADC).* Since it is not our intention here to dive into ADC details, we can describe it as just a black box, with two inputs and a few digital outputs, as shown in Fig. 2.7. One analog input is used for the voltage to be digitized, V_{in}, and another analog input expects a constant reference voltage, V_{ref}, that will define the input range. For example, an ADC with $V_{ref} = 8$ V can measure voltages in the range 0–8 V. An ADC has also N output pins, used to encode the result of digitization. The number of output bits N, also called *A/D resolution*, determines the number of alternatives that can be expected as output. For example, a 3-bit ADC can produce $2^3 = 8$ possible output values equal to 000, 001, 010, … or 111.

An ADC works as follows. It *samples* the analog continuous input voltage, and obtains a value V_{in}. Next, it compares this instantaneous reading V_{in} to the reference voltage V_{ref} and quantifies their ratio. Finally, the ADC generates the digital output, D, according to the formula:

$$D = \left[2^N \frac{V_{in}}{V_{ref}}\right] \tag{2.7}$$

where […] brackets mean truncation to N bits.

Figure 2.7 illustrates this mechanism for an ADC with a resolution of $N = 3$ bits and a reference voltage $V_{ref} = 8$ V. For example, if the input analog voltage is



Analog input	Digital output
0-1V	000
1-2V	001
2-3V	010
3-4V	011
4-5V	100
5-6V	101
6-7V	110
7-8V	111

Fig. 2.7 A block diagram and the input/output (I/O) mapping for an ADC with a resolution of $N = 3$ bits, and a reference voltage $V_{\text{ref}} = 8$ V

$V_{\text{in}} = 2.7$ V, then the ADC output D is $[8 \times 2.7/8] = [2.7]$, which when represented using 3 bits, becomes 010.

The effect of quantization is visible in Fig. 2.7 from the fact that for example, *any* input voltage between 1 and 2 V, will produce the same output, D = 001. This is due to the relatively low number of bits used for quantization, equal to 3 in our case, that creates a staircase effect in the output digital signal. The higher the number of output bits, or A/D resolution, the better the digital signal will resemble its analog "brother" at the input, as illustrated in Fig. 2.8.

The reverse process, of reassembling an analog signal from its digital samples, is called *reconstruction.* This happens in another electronic device, called *digital-to-analog converter (DAC)*. This process is equally essential for pervasive computing systems, in order to extract the right information carried by the analog signal and to send commands to different actuators, such as motors, displays, or loudspeakers.

If you managed to follow the story until now, you could figure out that once entered in a computer, a real-world continuous and analog signal gets robbed of its beautiful properties as effect of digitization. The effect can be, in some cases, catastrophic. Take a look at the 19 MHz signal in Fig. 2.9, sampled with a frequency $f_s = 20$ MHz. If we measure the period of the sampled signal, marked with red bullets, we get $T = 1$ µs, meaning that the signal's frequency is equal to $f = 1/T = 1$ MHz. But wait a minute, this is not the real frequency of our signal! Something must have gone wrong. Did magic formula (2.1) suddenly fail on us? No. The reason is that we took too few samples. This effect of undersampling is called *aliasing*. The false frequencies resulted after reconstruction are called *alias frequencies*.

This example should be a warning that digitization is a powerful technique, but also a dangerous one. When performed wrongly, it can create a "broken telephone"

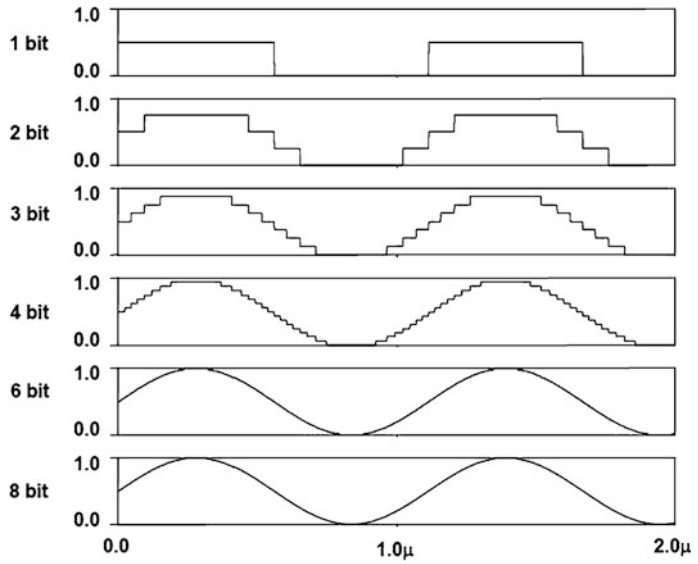


Fig. 2.8 Quantization of a 900 kHz sine wave with different resolution, varying from 1 bit to 8 bits. From [1]. Used with permission

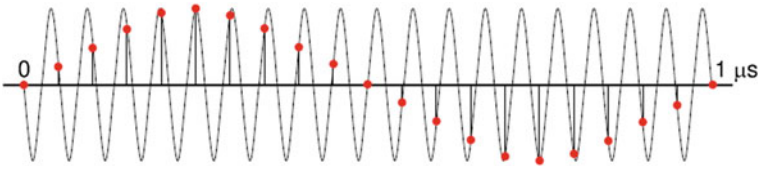


Fig. 2.9 A 19 MHz signal is sampled with a frequency f_s = 20 MHz. From [1]. Used with permission

effect, by corrupting the information carried by the signal. And this is the nightmare of any messenger, right? You might say: "OK, I understand the danger, but what can I do about it?" The solution is to perform a "good", careful digitization, that will distort the signal as little as possible. This can be realized by wisely controlling the digitization parameters, or more precisely, the sampling rate and the ADC resolution.

The proper ADC resolution is dictated by the smallest input variation that the system is required to "feel". In our example from Fig. 2.7, we saw that a 3-bit ADC with V_{ref} = 8 V was insensitive to input variation smaller than $V_{ref}/2^N$, in our case equal to 1 V. For some systems, this resolution limit might be acceptable. However, others, more sensitive systems, might require a higher resolution. Adding a few bits to the ADC resolution strongly improves the performance of quantization. For example, an 8 bits ADC can sense a minimal change in its input voltage of 8 V/2^8, equal to 0.03 V.

What of the ideal sampling frequency? This parameter depends on the frequency of the signal being measured. The goal is to obtain enough samples for an acceptable reconstruction of the signal, without too much information loss. But what means "acceptable"? For example, for a smart home, it might be acceptable to measure the room temperature once in a minute. A safety critical system, however, like an airbag control system, might need to measure the car's acceleration much more often, say one hundred times in a second. To ensure that there will be enough samples for reconstruction, it is required that the sampling period should not be greater that one half of the finest detail in the signal. This is known as the *Nyquist criterion* or the *Shannon sampling theorem* saying that:

A continuous analog signal s(t) with frequencies no higher than f_{max} *can be reconstructed exactly from its samples* $s[n] = s(nT_s)$*, if the samples are taken at a rate* f_s*, that is greater than* $2f_{max}$.

This means in fact that the sampling frequency has to be *at least* twice the maximum frequency in the signal. This minimal sampling frequency given by $2f_{max}$ is called the *Nyquist rate*.

Let us consider, for illustration, the signal in Fig. 2.10 built by adding together three sinusoidal waves at frequencies of 1, 2, and 3 Hz. According to the Nyquist criterion, the signal must be sampled at twice its highest frequency. This highest frequency is in our case $f_{max} = 3$ Hz, meaning that a sampling frequency of 6 Hz (6 samples per second) should be just enough. However, for a good reconstruction, it is recommended to use a sampling frequency that is ten times higher than f_{max}.

Some best practice sampling rates commonly used are: 8 kHz, 8–12 bits for telephony, and 14–16 bits for mobile phones; 44.1 kHz, 16–24 bits for CD audio players; 8 kHz up to 20 kHz for analog speech signals, what makes sense, because 20 kHz is the generally accepted upper limit for human hearing and perception of musical sounds.

You might now ask a naughty question: "What happens if we do not obey the Nyquist criterion?" Sampling with a higher rate than the Nyquist rate, called *oversampling*, is not as bad. It will even improve the situation, creating a smoother

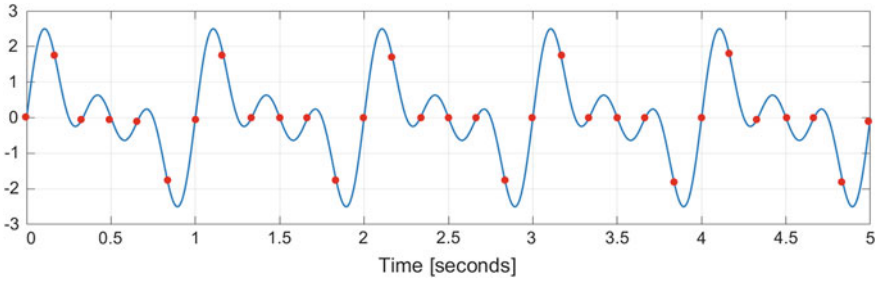


Fig. 2.10 Sampling with $f_s = 6$ Hz the sum of three sine waves of 1, 2, 3 Hz

reconstructed waveform. However, be careful with overdoing it, because a high sampling frequency means by definition more samples, that boosts up the storage and computational needs of the system. What might happen in case of *under-sampling* with a sampling frequency lower than the Nyquist rate, is known as aliasing, a phenomenon already demonstrated earlier in this section.

If we know the A/D resolution and sampling rate that are considered as acceptable for our system, we can choose an ADC device. The choice in doing this is large. ADCs are available for different market segments, with a range of resolutions varying from 8 to 24 bits, and maximum sampling rates between 100 Hz and 5 GHz. The choice is dictated by a trade-off between the application's required performance and available resources (memory, budget, etc.).

2.2.2 Two-Dimensional Digital Signals

We already know quite a lot about one-dimensional time signals, and how they are dealt with in pervasive computing systems. But how is the life of two-dimensional signals, such as 2D images? These signals are also continuous and analog, and therefore also need to be digitized. This again involves *sampling* and *quantization*. First, a spatial sampling takes place that converts the image into a 2D structure, where the brightness is known only at discrete positions in space. These discrete points in 2D space are called *pixels*, a mnemonic for "picture elements". If we denote by M an image size (number of pixels) in the horizontal direction, and by N the image size in the vertical direction, then we say that the *spatial resolution* of the image is $M \times N$. In practice, typical spatial resolution values are 640×380, 1072×640, etc. A high spatial resolution means that a large number of pixels are used, resulting in fine details in the image. Similar to the unidimensional case explained in the previous section, a low number of pixels results in a "blocky" effect in the image. The visual effect of different spatial resolutions can be seen in Fig. 2.11.

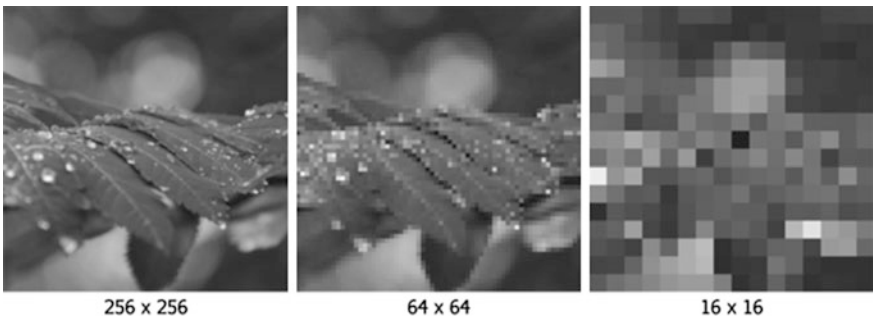


Fig. 2.11 Effect of spatial resolution. A 256×256 image has a higher resolution than a 16×16 one. From [2]. Used with permission

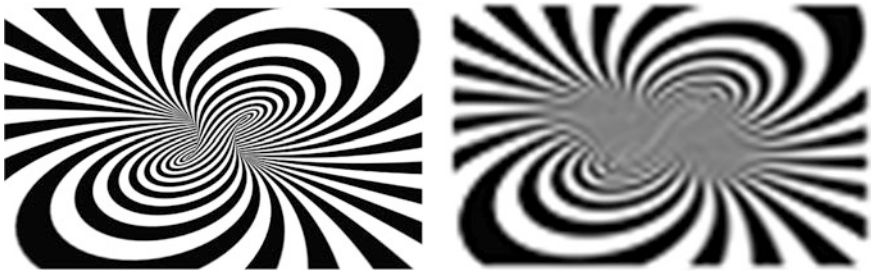


Fig. 2.12 Effects of spatial undersampling. Inspired from [3]

The same as with time signals, spatial oversampling is benefic and results in a higher quality image. However, huge amounts of data are created, that need to be processed and stored. Spatial undersampling, however, might create aliasing and false features, like the jagged edges in Fig. 2.12.

We are not done yet, because the pixel's brightness or color can be stored in a computer by using only a limited number of bits. Therefore, spatial sampling has to be followed by a quantization of the pixel color. The number of bits necessary to represent the color of a pixel is called *color resolution*. A higher color resolution results in a higher color quality.

The result of sampling followed by quantization, in one word known as digitization, is a digital image, carrying discrete-space and discrete-color information. A digital 2D image is mathematically represented by a two-dimensional discrete-space, discrete-color function that assigns to each pixel a color. A digital 2D image is stored in a computer as a two-dimensional array, denoted by p. Each element of this array, $p[x, y]$, stores the color of the pixel situated in position (x, y). This color is a finite-precision integer number.

Images can be stored in a computer in different formats, each having a specific color resolution (see Fig. 2.13). The simplest one is the *binary,* or *black-and-white* (*B&W*) format, where each pixel can be just black or white. The color resolution is in this case only one bit. Obviously, this is the most compact format that requires the least amount of memory space. A 100×100 binary image, for example, will need only $100 \times 100 = 10{,}000$ bits of memory storage.

In a *grayscale image,* each color is a shade of gray, varying usually from 0 (black) up to 255 (white). The color resolution is in this format 8 bits. A 100×100 grayscale image will need therefore more memory space, namely $100 \times 100 \times 8 = 80{,}000$ bits.

In a *true color,* or *red–green–blue* (RGB) image, each pixel has a color described by the amount of red (R), green (G), and blue (B) in it. Each of these three components is a value in the range of [0–255], giving us $2^8 \times 2^8 \times 2^8 = 2^{24} = 16{,}777{,}216$ possible color alternatives for each pixel. For example, a black

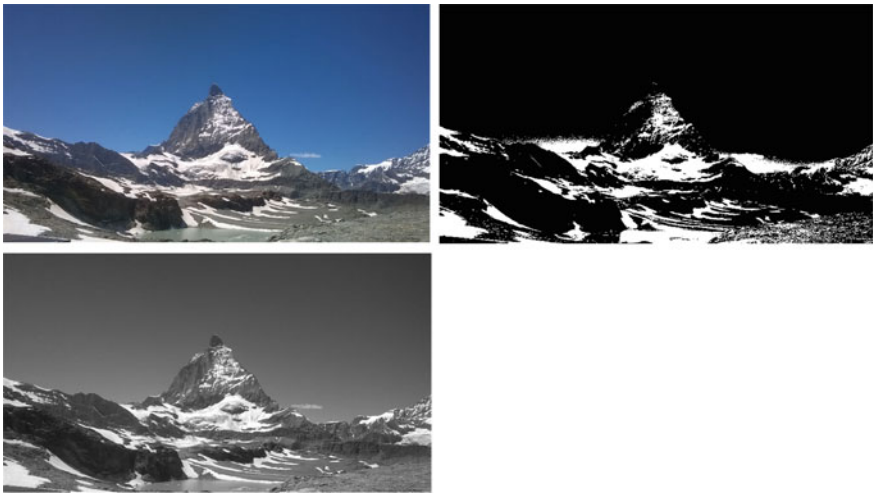


Fig. 2.13 Examples of digital image formats. An RGB color image, *upper left corner*; the same image in grayscale format *left under*; the same image in binary format, *upper right* corner

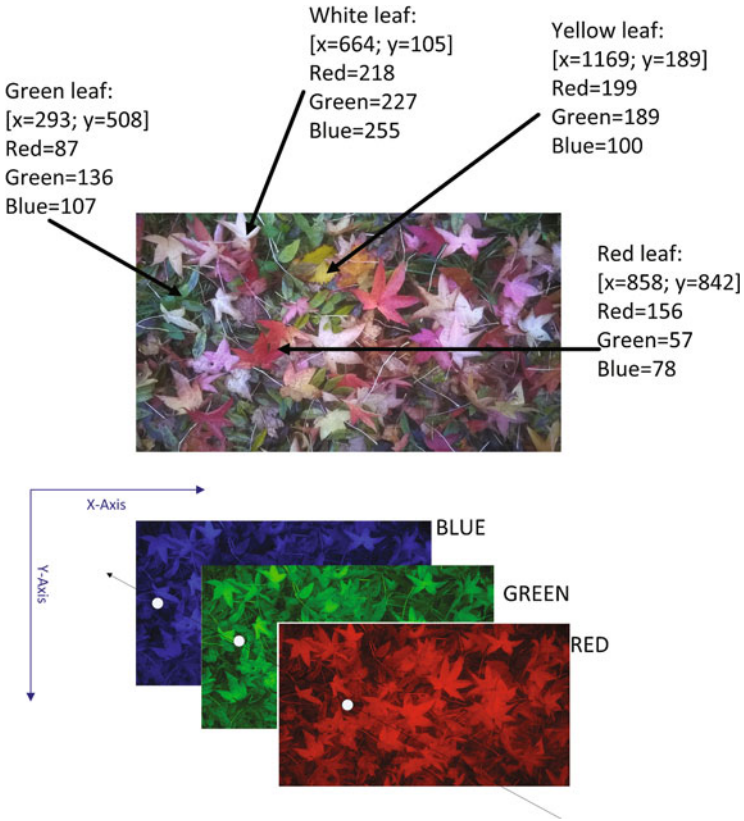


Fig. 2.14 Color decomposition of an RGB image

pixel is represented as (0, 0, 0), a pure red pixel is (255, 0, 0), and a white one is represented as (255, 255, 255). In Fig. 2.14, the pixel on the red leaf has a red value that is higher than the green and blue components. Likewise, the pixel on the green leaf has a higher green component. The color resolution is in this format equal to $3 \times 8 = 24$ bits. Images stored in this format are therefore also called 24-*bit color images*. Their storage requires also much more memory space. The 100×100 RGB image needs $100 \times 100 \times 24 = 240{,}000$ bits!

An $M \times N$ RGB color image is internally represented with three $M \times N$ arrays, one for each of the colors red, green, and blue. You can imagine an RGB image as a three-colored sandwich, made of three equally sized $M \times N$ slices of bread, containing Red, Blue, and Green information, respectively. The color of each pixel p[x, y] in the image is represented by a vector with three elements, p[x, y] = [Red (x, y), Green(x, y), Blue(x, y)]. To determine the values of these three elements, one can insert an imaginary toothpick through the three slices, at that selected position (x, y) and read the pure colors in the three punctures. These will give the three RGB values that together characterize the pixel's color, like illustrated in Fig. 2.14.

2.3 Conclusions

This chapter described the life of real signals, in their role of information messengers traveling between environment and pervasive computing systems. The systems use sensors to capture a signal from the environment and transform it into an electrical signal having the same pattern of variation. This electrical signal is continuous and analog, properties that are in conflict with the discrete, limited precision character of a digital computer. The solution for this conflict is a process called digitization. Digitization happens in two stages: sampling and quantization. If an analog continuous signal can be represented with a function of continuous domain and range, then sampling means discretization of the function's domain, and quantization means discretization of its range. Digitization is the task of an ADC. The result is a sequence of digital numbers, called samples. These samples are processed by the software controller and contribute to the making of appropriate decisions. These decisions must be translated in commands to actuators. Actuators, by nature, are devices that expect an analog electrical input signal. Therefore, before leaving the system, a signal reconstruction from its samples is needed, performed by a DAC. Digitization and reconstruction are processes of paramount importance for the quality of a pervasive computing system. Therefore, both must be performed with care, with minimal loss of information. The block diagram in Fig. 2.15 illustrates the findings of this chapter.

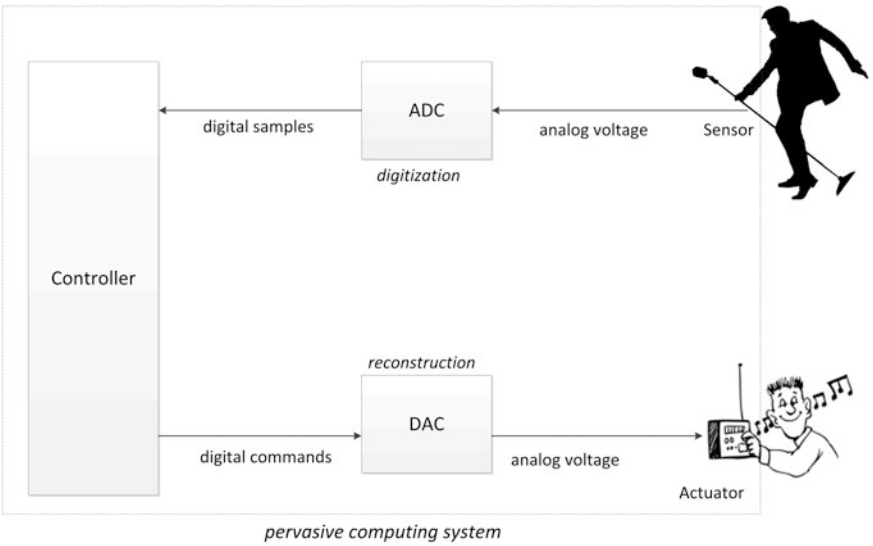


Fig. 2.15 A block diagram illustrating the life of signals in a pervasive computing system

2.4 Exercises

1. Draw a careful sketch for each of the following signals.
   (a) a sinusoidal signal with a DC component of 5 V, frequency $f = 0$ Hz, and amplitude $A = 5$ V (volts).
   (b) a sinusoidal signal with no DC component, the amplitude $A = 2$ V, frequency $f = 2$ Hz, and phase shift $\varphi = 0$.
   (c) a sinusoidal signal with no DC component, the amplitude $A = 2$ V, frequency $f = 0.2$ Hz, and phase shift $\varphi = 0$.
   (d) the sum of the signal from (a) and the signal from (b).
2. Figure 2.16 shows a plot of a sinusoidal waveform.

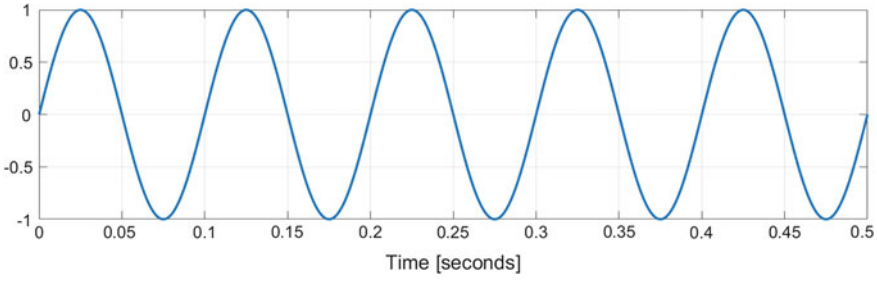


Fig. 2.16 A sinusoidal waveform

From this plot, determine its amplitude, frequency, and phase shift. Give the answers as numerical values and units. Justify your answers.

3. What sampling frequency would you recommend for the signal from Exercise 2? Justify your answer. How many samples will we get?
4. Imagine that the signal from Exercise 2 is sampled with a frequency of 100 Hz, for two periods. How many samples do we get and what is the time step between two consecutive measurements? Can you mark the samples on the plot?
5. What is an ADC, and why do we need it in pervasive computing? Answer the same question for a DAC.
6. Calculate the maximal quantization error for the ADC in Fig. 2.7. Make a new I/O mapping for a better ADC, with 8 bits and $V_{\text{ref}} = 8$ V. Show how a higher resolution improves quantization.
7. The note A above the middle C has a frequency of 440 Hz. What does the Shannon sampling theorem say about the frequency that we have to use to sample this sound properly? What will happen if we do not obey this criterion?
8. Define the concepts of spatial resolution and color resolution for a digital image. Give one example for each concept.
9. Explain how colors are represented in an RGB digital image.

References

1. Pelgrom, M.: Analog-to-Digital Conversion. (2010)
2. Moeslund, T.B.: Introduction to Video and Image Processing: building real systems and applications. Undergraduate Topics in Computer Science. (2012)
3. McAndrew, A.: Elementary Image Processing with Matlab. (2004)

http://www.springer.com/978-3-319-51654-7

Pervasive Computing
Engineering Smart Systems
Silvis-Cividjian, N.
2017, XVIII, 210 p. 183 illus., 99 illus. in color., Softcover
ISBN: 978-3-319-51654-7