3 DSP Basics

Jason Alexander and Chi Thanh Vi

3.1 Introduction

This chapter introduces the fundamentals of Digital Signal Processing (DSP) necessary for the design and construction of any interactive system that directly engages with the environment using sensors. There is a diverse range of such systems, including capturing the speech input for human-computer conversations [Cassell et al. 2000], hand gestures to input text [Jones et al. 2010], eye-gaze to facilitate more efficient interaction [Pfeuffer et al. 2015], recording biosignals such as heart rate to understand the body [Schmidt 2015], adapting digital content based on environmental conditions or locations [Rodden et al. 1998], and in each of the Case Studies described later in this book. All must capture and then *process* sensor data in order to provide input, or adapt output in a digital system. In many cases, the interactive systems developer must convert, filter, transform, and/or threshold raw signal data (from e.g. an accelerometer, microphone, or ECG sensor) into a form suitable for use in their application. The choices made in each Digital Signal Processing step have implications on the quality of the resulting output used to direct the decisions made by their application.

To get the reader started, we aim to provide a beginners-guide to DSP. However, this introduction is far from exhaustive. Indeed, there are numerous great textbooks dedicated to advanced understanding of this topic and the reader is encouraged to consult them for more in-depth theoretical knowledge. Recommended reading includes the 'Digital Signal Processing' titles by Proakis and Manolakis [2007] and Rawat [2015].

DSP basics begins with an introduction to the different types of signals, and continues to cover the analog-to-digital conversion topics of sampling, quantization, and coding. From there it covers digital-to-analog conversion, Discrete Fourier Transforms, autocorrelation, and Linear Time-invariant Systems. Finally, it ends with a walk-through of DSP use in Brain-Computer Interaction (BCI).

3.2 Signals

Proakis and Manolakis [2007] define a *signal* as "any physical quantity that varies with time, space, or any other independent variable or variables". In interactive systems, input signals may derive from one of many possible sources, including [Wilson 2004]:

Audio: Audio signals are a representation of sound, often consisting of a range of different frequency acoustic waves. Microphones are used to capture audio signals, typically for transmission, recording, or analysis. Such audio capture is often subject to background noise that requires filtering. The most important frequencies for capture and analysis are those in the range of human hearing: 20 Hz–20 kHz.

Biosignals: A biosignal is any signal present in a living being, both electrical and non-electrical. These can be used to understand the health, activity, emotional state, or intention of the monitored being, typically a human in interactive systems development. Examples include Electroencephalogram (EEG; electrical activity of the brain), Electrocardiogram (ECG; activity of the heart), Electromyogram (EMG; activity of skeletal muscles) and Electrodermal Activity (EDA; sweat gland activity on the skin). Interest in biosignal analysis has grown significantly with the rise of digital health research and relevant enabling technologies [Schmidt 2015, Silva et al. 2015].

Chemical: Chemical sensing is used to detect the presence or concentration of a specific chemical compound or element. There are a wide array of molecule sensing techniques including: use of their vibrational and rotational properties, their mass spectra, through chromatography (separation of a mixture), electrochemical sensing (difference in electrical charge across a membrane), liquid conductivity, or by measuring charge transfer in solution or gas with a CHEMFET (charge transfer across the surface of a Field Effect Transistor). Chemical sensing has a wide range of applications from automotive (monitoring the correct fuel-to-air ratios are used for combustion) to water purity sensing.

Displacement: Position sensors detect a target object, person, or disturbance in an electrical or magnetic field and convert that into an electrical output. Contact position sensors use limit switches or resistive position transducers to convert physical position into an electrical signal; non-contact position sensors include magnetic and magneto-resistive sensors, ultrasonic sensors, photo-electric sensors, capacitive, and inductive sensors. Position sensors are widely used in interactive systems, from capacitive touch screens [Grosse-Puppendahl et al. 2017, Zimmerman et al. 1995], to 3D gesture sensing [Suarez and Murphy 2012], to tangible interaction [Taher et al. 2015]. Some displacement sensors are also used to measure physical movement.

Environmental: Capturing the state of an environment may include variables such as the temperature, humidity, air pressure, and ambient light. Temperature is the amount of heat energy in a system and is measured either through contact or non-contact sensing. Humidity is the water vapour content in air; humidity sensors are now cheap and accurate and are typically capacitive, resistive, or use thermal conductivity. Environmental sensing is typically used to understand environmental patterns and to inform decisions on heating, air-conditioning, and/or lighting.

Physical Movement: Any physical movement of a inanimate object or living being is subject to acceleration, which can be measured (along a single axis) by an accelerometer.

Accelerometers have wide application in interactive systems, capturing acceleration, shock, and vibration and can be installed in devices from airbag sensors to in-phone step counters or pedometers. The most common form of accelerometers are based on the piezoelectric effect: a mass is placed on a piezoelectric material, when the mass is accelerated it applies a force on the material which in-turn generates an electrical current proportional to the mass and acceleration. Examples of their use in interactive systems include for gestural input [Jones et al. 2010], health behaviour capture [Klasnja et al. 2011], and musical instruments [Kiefer et al. 2008].

Accelerometers can measure linear acceleration, however they are unable to measure twisting or rotational movement. Instead, a gyroscope can measure angular velocity along three axes: pitch (x axis), roll (y axis) and yaw (z axis); this allows it to determine an object's orientation in 3D space. Finally, magnetometers can be used to find the earth's magnetic north. An Inertial Measurement Unit (IMU) combines accelerometers, gyroscopes, and magnetometers to measure acceleration, angular velocity, and magnetic fields; their outputs can be combined to determine motion, orientation, and heading.

Visual: Visual signals can range from simple optical photo-sensors (detecting the presence or absence of light) to full-colour images captured through CCD (Charged-Couple Device) or CMOS (Complementary Metal-Oxide Semiconductor) sensors. In most cases, an analog photo-signal is converted into a digital representation for later analysis or reproduction.

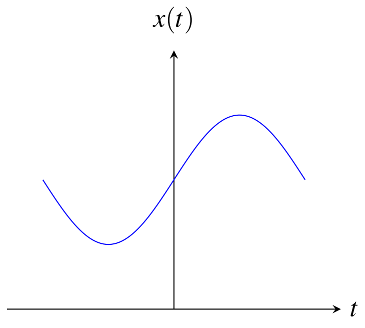
All signals carry information; *signal processing* is the process of extracting the useful information carried by the signal. Most signals sensed from the real world are analog; we cover this conversion to digital form later in this chapter. First, it is important to understand the different types of signals and their properties.

3.2.1 Classification of Signals

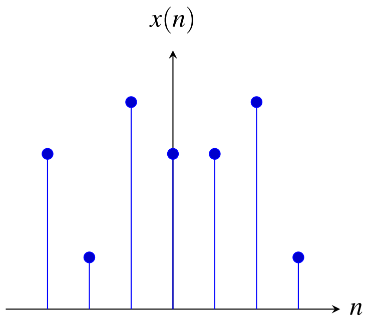
There are a wide variety of signal types; understanding the difference between these will help the practitioner to correctly interpret and analyse a diverse range of signals. Each must be treated differently based on their properties. Below we cover the most important signal types for interactive systems development.

3.2.1.1 Continuous-time vs. Discrete-time Signals

The difference between continuous- and discrete-time signals is illustrated in Figure 3.1. Figure 3.1a shows a continuous-time (or analog) signal where there is a continuum of values for the independent variable, x, for all values of time, t. Conversely, Figure 3.1b shows a discrete-time signal, where values of x are only defined for specific values of n. The amplitude (value) of the discrete-time signal between two defined time points, x_n and x_{n+1} is not defined.



(a) Continuous-time signal



(b) Discrete time signal

Figure 3.1: Visual representations of continuous- and discrete-time signals.

In interactive systems, analog sensors produce continuous-time signals, for example those that capture light intensity, sound, or atmospheric pressure. Conversely, discrete-time signals are generated by digital sensors such as electro-optical rotary encoders (for measuring motion) and image sensors (for capturing still or moving frames). Many analog sensors are now also available in their digital equivalents—the choice between them will depend on application, price, and required signal type.

3.2.1.2 Continuous-value vs. Discrete-value Signals

The value of any continuous-time or discrete-time signal can itself be continuous or discrete. When a signal can take on all possible finite or infinite values of a range it is a *continuous-value* signal. When a signal can only take on a discrete number of values, then it is a *discrete-value* signal. A discrete-time and discrete-valued signal is called a digital signal.

3.2.1.3 Multi-channel and Multi-dimensional Signals

In some interactive systems, signals may be generated from multiple sensors simultaneously—for example, in EEG, where multiple electrodes monitor electrical activity on the skull. Such *multi-channel* signals can be represented as a vector:

$$x(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{pmatrix}$$

Where $x_k(t)$ represents the signal from the kth sensor at time t. If a signal is formed of a single source independent variable, then it is called *one dimensional*; if it is formed from M different independent variables then it is *M-dimensional*.

3.2.1.4 Deterministic and Random Signals

Deterministic signals can be described completely by a mathematical formula from which the amplitude of the signal at any given time is calculated. Conversely, signals whose amplitude cannot be mathematically calculated are *random*. They may instead be described by a probabilistic description (e.g. the mean value).

3.2.1.5 Periodic and Aperiodic Signals

Periodic signals are those that continually repeat their amplitude pattern within a measurable time frame. A time-signal, $f(x)$, is periodic, if and only if:

$$f(x) = f(x+T) \text{ for all } x.$$

Where T is any positive integer. The completion of a full pattern is called a *cycle*, while the *period* of a signal, T, is the time in seconds required to complete one full cycle. The amplitude pattern of a periodic signal remains unchanged when it is time-shifted by one-period. Conversely, functions that never repeat themselves have an infinite period and are known as *aperiodic* signals.

3.3 Analog-to-Digital Conversion

Many of the input signals to interactive systems are analog signals captured from the real world (e.g. heart rate, accelerometer, force etc). In order to process these signals digitally, they must first be converted into a digital form. This process is called *analog-to-digital conversion* and is generally used to digitize signals that vary in voltage. There are three primary steps in the conversion process (see Figure 3.2) [Proakis and Manolakis 2007]:

1. **Sampling:** Conversion of a continuous-time signal into a discrete-time signal by taking "samples" at instants in time.
2. **Quantization:** Conversion of the discrete-time continuous-valued signal obtained by 'sampling' into a discrete-time, discrete-valued (digital) signal.
3. **Coding:** Each discrete value is converted into a binary representation.

The remainder of this section describes this process along with the key steps and limitations.

3.3.1 Analog Signal Sampling

Signal sampling is the process of acquiring the instantaneous amplitude (magnitude or value) of a signal at a particular point in time. In interactive systems, *periodic* or *uniform sampling* most often occurs where the amplitude is sampled at regular intervals. This is visually illustrated in Figure 3.3. This type of sampling can be described by [Proakis and Manolakis 2007]:

$$x(n) = x_a(nT), \qquad -\infty < n < \infty$$

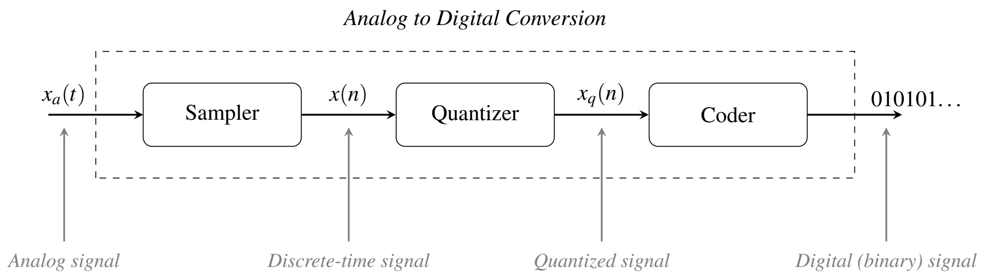
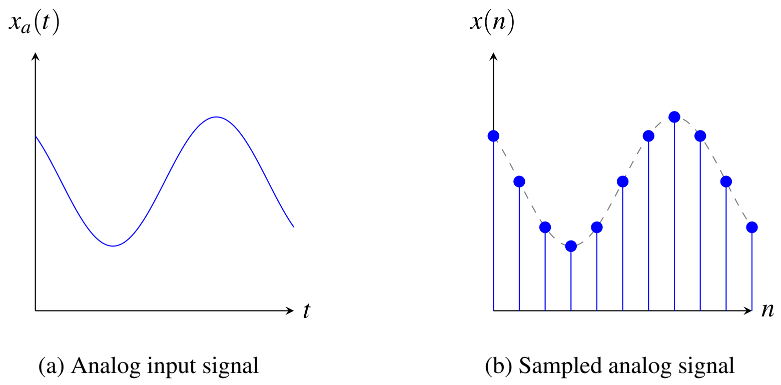


Figure 3.2: The analog-to-digital conversion process.



(a) Analog input signal (b) Sampled analog signal

Figure 3.3: Uniform sampling of an analog signal.

Where $x(n)$ is the discrete-time signal obtained by sampling the analog signal, $x_a(t)$, every T seconds. The time between successful samples is called the *sampling period* and its reciprocal, is the *sampling frequency*:

$$F_s = \frac{1}{T},$$

which is measured in samples per second (Hz).

3.3.2 Aliasing and Anti-Aliasing

Aliasing occurs when an analog input signal is not sampled at a sufficiently high-enough frequency to recover the original analog signal. An example of this is shown in Figure 3.4. The blue input signal is sampled once every second (1 Hz, black dots). This is insufficient to reproduce the original input signal and instead we reproduce the signal shown in red. The red

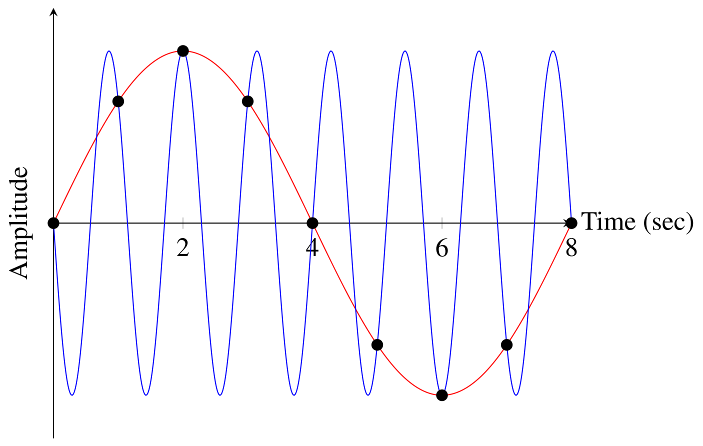


Figure 3.4: Sampling the blue signal (-$\frac{7}{8}$ Hz) at too low a rate (1 Hz, black dots) results in signal aliasing (red signal).

signal is known as an *alias* of the blue input signal as we have insufficient sampling points to distinguish the two signals.

To overcome this, we can use *Nyquist-Shannon Sampling Theorem* to determine the minimum required sampling rate for an input signal in order to later fully recover that signal. The theorem states that the sampling rate (f_s) must be at least twice as large as the highest frequency component (f_{max}) of the analog input signal. More formally, this is written as:

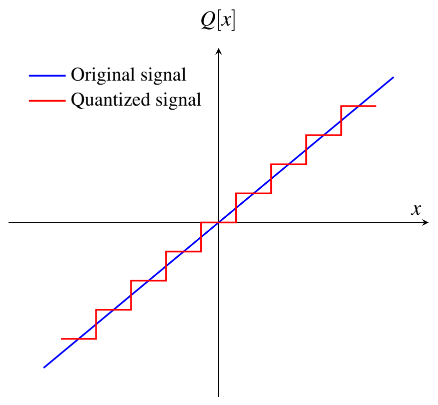
$$f_s \geq 2f_{max}$$

This minimum sampling frequency is referred to as the Nyquist frequency.

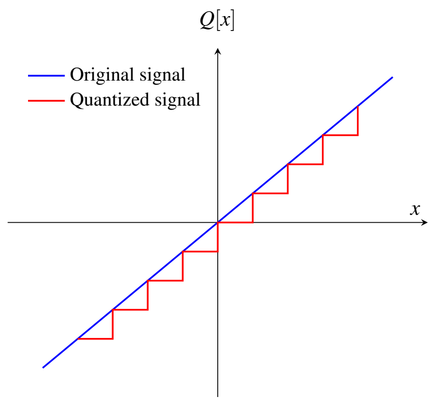
Example: In order to reconstruct a speech signal containing frequencies upto 4 kHz, the minimum sampling rate is 8 kHz or 8,000 samples per second.

3.3.3 Quantization

Quantization is the process of converting a discrete-time, continuous-amplitude signal into a discrete-time, discrete-amplitude signal. Signal sampling selects the discrete points in time where we will record signal information; quantization takes the values of those discrete points and maps (or constrains) them onto a discrete set of digitally-representable values. This involves rounding and truncation as computing systems are unable to represent values to an infinite precision. After quantization, the signal becomes a *digital signal*. The information lost in the quantization process can never be recovered.



(a) Quantization using rounding



(b) Quantization using truncation

Figure 3.5: Visualization of rounding and truncation quantization techniques

There are two primary schemes for signal quantization: (1) scalar quantization, where each signal sample is treated individually and; (2) vector quantization, where blocks of the signal samples are processed together. Scalar quantization schemes are uniform or non-uniform: uniform quantization has levels of equal length, non-uniform does not. We describe the two types of scalar quantization in this chapter.

3.3.3.1 Uniform Scalar Quantization

Scalar quantization schemes are either uniform or non-uniform. Both operate to reduce a signal's amplitude to a finite number of *quantization levels* (L). Uniform quantization schemes use evenly spaced quantization levels; therefore a quantizer that uses b bits can represent $L = 2^b$ different levels. The spacing between these levels is known as *quantum*, *step-size*, or *resolutions* (Δ). Given R is the peak-to-peak range of the input signal $x(n)$, the step-size available can be calculated as:

$$\Delta = \frac{x_{max} - x_{min}}{L} = \frac{R}{L}$$

A signal sample's value can be quantized by rounding to the nearest quantization level or by always truncating downwards. The difference these two techniques make to the quantized signal can be seen in Figure 3.5. Quantized signals are stored in *words* of a specified finite number of bits. This finite word length leads to a limit on the number of quantization levels possible and therefore undesirable representation effects or *finite-word-length effects*.

3.3.3.2 Quantization Error and Quantization Noise

The finite-word-length effect inevitably leads to some mis-representation of the input signal; this is called *quantization error* or *quantization distortion*, $e(n)$. This is quantified as the difference between the unquantized value, $x(n)$, and the quantized value $Q[x(n)] = x_q(n) = \hat{x}(n)$ and can be expressed as:

$$e(n) = Q[x(n)] - x(n) = x_q(n) - x(n)$$

For quantization by rounding, the quantization error can be described as:

$$-\frac{\Delta}{2} < e(n) \leq \frac{\Delta}{2}$$

When the number of quantization levels, L, is large, and therefore the step-size, Δ, is small, the error is equally likely to be uniformly distributed between $-\frac{\Delta}{2}$ and $\frac{\Delta}{2}$. For this reason, quantization error is often called quantization noise.

3.3.3.3 Non-Uniform Scalar Quantization

A non-uniform quantizer distributes the quantization levels in proportion to the Probability Distribution Function (PDF) of the input signal. The requirement that the quantization level size is constant for all levels no longer applies. This can significantly improve the performance of the quantizer. When the PDF is non-uniform, finer quantization (i.e. a higher number of representation levels) should be applied when the PDF is large and coarser quantization (fewer quantization levels) should be used where the PDF is small. Determining the mapping of PDF to quantization level size is beyond the scope this chapter.

3.3.4 Coding of Quantized Samples

Finally, the coding process assigns a unique binary value to each quantization level. If there are L quantization levels, then there must also be L different binary values available. A word length of b bits can represent 2^b different binary numbers. The following equation can be used to calculate the number of bits required for L levels:

$$b \geq log_2 L$$

The numbers of bits, b, must be an integer value. Typically, commercial analog-to-digital converters have a fixed precision of up to b = 12.

3.3.5 Analog-to-Digital Converters (ADCs)

The three elements of Analog-to-Digital conversion (sampling, quantization, and coding) are performed by Analog-to-Digital converters (ADCs). There are several approaches to constructing an ADC converter. We will briefly examine two in this section: Flash ADCs and Successive Approximation Converters.

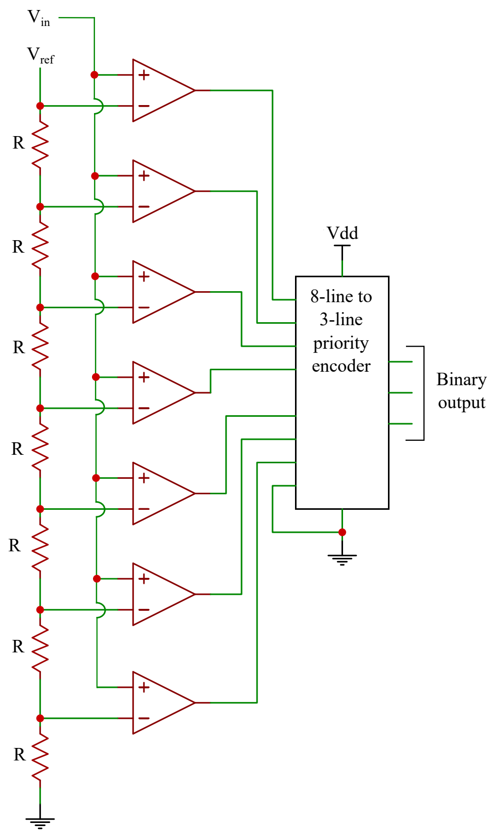


Figure 3.6: Example circuit for a Flash ADC.

3.3.5.1 Flash ADCs

Flash ADCs are the fastest converters; they use a series of comparators (op-amps) to perform ‘parallal’ analog-to-digital conversion (see Figure 3.6). Each comparator compares V_{in} to a different reference voltage. While Flash ADCs are very fast, they are also expensive, as with each additional output bit, the number of required comparators doubles.

3.3.5.2 Successive Approximation ADC

Successive Approximation ADCs (SARs) use a comparator and counting logic to perform the analog-to-digital conversion, successively deciding whether the input is above or below a narrowing range’s midpoint. For example, given a 5V input signal and a reference of 8V, we find that 5V is above the midpoint of 0–8V (the midpoint is 4V) and so the Most Significant Bit (MSB) of the output is set. We then compare the 5V input to the range of 4–8V. In this

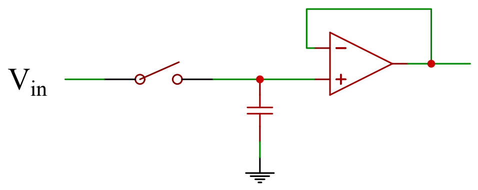


Figure 3.7: Example sample-and-hold circuit.

case, 5V is below the midpoint of this range, meaning the next bit is not set. This process continues until all bits are resolved. Given their serial nature, SARs are considerably slower than Flash ADCs.

3.3.6 Sample and Hold Circuits

During analog-to-digital conversion the input signal must be maintained at a constant value, especially in conversions when it is sampled multiple times (e.g. in a SAR). To do this, the input signal is often sourced through a Sample-and-Hold circuit—this 'grabs' an analog sample and holds that value steady for at least the time required for the conversion. This prevents errors during the conversion process. An example of such a circuit is shown in Figure 3.7.

3.3.7 Analog-to-Digital Conversion in Interactive Systems Prototyping

Interactive Systems prototyping often involves popular micro-controller platforms such as the Arduino [Arduino AG 2018a], Raspberry Pi [Raspberry Pi Foundation 2018], or the BBC micro:bit [Micro:bit Educational Foundation 2018]. This section examines the analog-to-digital conversion capabilities of these devices and practical considerations when building interactive prototypes with such platforms.

The Arduino Uno (revision 3) [Arduino AG 2018b] is based on the Atmel ATMega328P chip whose package contains a successive approximation Analog-to-Digital Converter (ADC). This has a 10-bit resolution, meaning it can represent $2^{10} = 1024$ unique levels, with conversions taking between $13\,\mu s$ to $260\,\mu s$ [Microchip Technology Inc. 2018]. For most regular interactive systems applications, such as measuring the temperature or force, or capturing data from body-worn accelerometers for gestural input this resolution proves sufficient. However, for high-resolution applications, such as high-quality audio capture, developers should consider a stand-alone, higher-resolution ADC.

The BBC micro:bit is based around the Nordic Semiconductor nRF51822 chip that also contains a packaged ADC chip. Similar to the Arduino, this has up to 10-bit resolution,

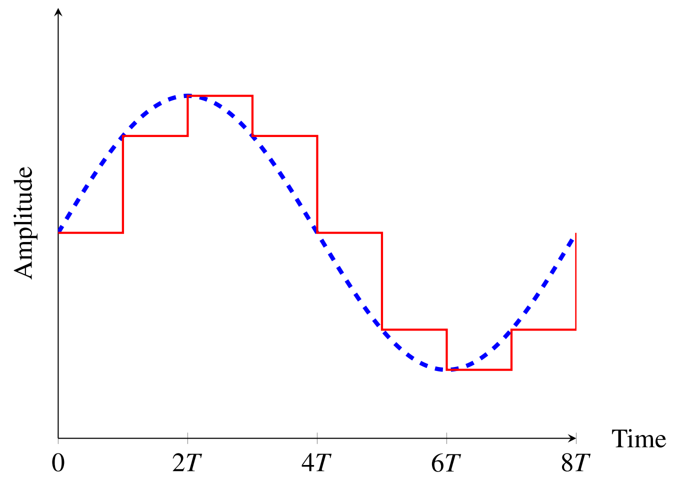


Figure 3.8: Zero-order hold (staircase) digital (red line) to analog (blue line) conversion.

with conversions taking between 20 µs to 68 µs, depending on the resolution (8, 9, or 10-bit) [Nordic Semiconductor 2018].

The Raspberry Pi contains no internal ADC functionality. However, common ADC chips, such as the MCP3008 provide equivalent functionality. These chips also have a 10-bit resolution and a maximum sampling rate of 200 ksps. The digital output is communicated back to the Pi using the SPI standard.

3.4 Digital-to-Analog Conversion

Digital-to-analog conversion performs the reverse operation: taking a digital (binary) signal and producing an analog output. This process allows humans to, for example, hear digitally-transmitted speech or audio. A Digital-to-Analog Converter (DAC) interpolates values in-between the given input samples. Figure 3.8 illustrates the simplest form of digital-to-analog conversion, *zero-order hold* or *staircase* approximation. Other approaches are also possible, including *linear interpolation* (linearly connecting successive pairs of samples), and *quadratic interpolation* (fitting a quadratic through three successive samples). More complex conversion approximations generally have a higher cost of implementation (either in hardware or computation time).

The simplest implementation of a DAC is the R-2R ladder (see Figure 3.9). The R-2R ladder works on the principle of superposition—switching on binary inputs adds more voltage at the output. R-2R ladders are inexpensive and easy to manufacture as they only require two resistor values.

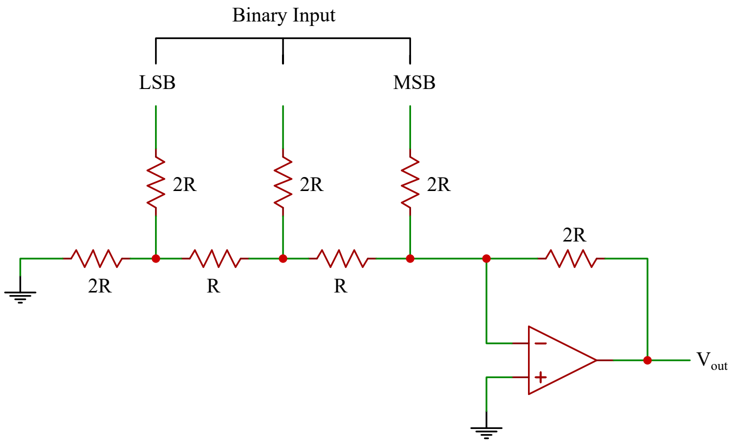


Figure 3.9: 3-bit R-2R ladder DAC.

3.5 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform (DFT) is one of the most important tools in Digital Signal Processing [Smith 2002]. There are three common applications of DFT:

Calculate a signal's frequency spectrum: This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding.

Find a system's frequency response: Find a system's frequency response from the system's impulse response, and vice versa. This allows systems to be analysed in the frequency domain, just as convolution allows systems to be analysed in the time domain.

Intermediate step: Be an intermediate step in more elaborate signal processing techniques. The classic example of this is FFT convolution, an algorithm for convolving signals (explained in section 3.7.1) that is hundreds of times faster than conventional methods.

The Fourier Transform of an original signal, $x(t)$, is:

$$X_k = \int_{-\infty}^{\infty} x(t)e^{-kt}dt$$

The Discrete Fourier Transform (DFT) is the equivalent of the continuous Fourier Transform for signals known only at N instants by sample times T (i.e. a finite sequence of data). DFT transforms this sequence of N complex numbers (instants) $\{x_n\} := x_0, x_1, ..., x_{N-1}$ into another sequence of complex numbers $\{X_k\} := X_0, X_1, ..., X_{N-1}$, which is defined by:

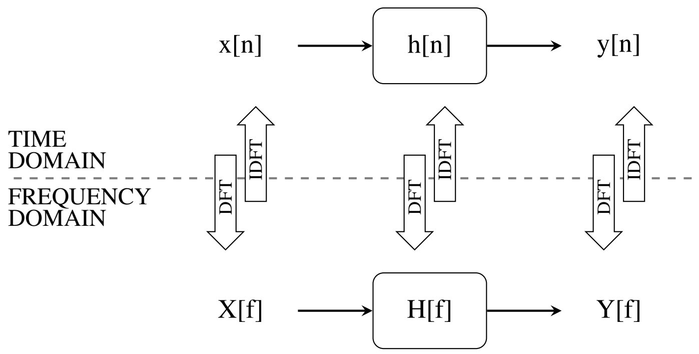


Figure 3.10: System operations in the time and frequency domains. In the time domain, an input signal ($x(n)$) is convolved with an impulse response ($h[n]$), resulting in an output signal ($y[n] = x[n] * h[n].$). In the frequency domain, an input spectrum ($X[f]$) is multiplied by a frequency response ($H[f]$), resulting in an output spectrum ($Y[f] = X[f]xH[f]$).

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi ikn}{N}} \\ &= \sum_{n=0}^{N-1} x_n \cdot [cos(2\pi kn/N) - i \cdot sin(2\pi kn/N)] \end{aligned}$$

for $0 \le k \le N-1$. The effect of computing the X_k is to find the coefficients of an approximation of the signal by a linear combination of sinusoidal waves. Since each wave has an integer number of cycles per N time units, the approximation will be periodic with period N. This approximation is given by the Inverse Fourier Transform (IFT):

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{2\pi ikn/N}$$

Figure 3.10 illustrates the differences in system operations between the two domains: time and frequency.

Example 1: let $x_n = \{x_0, x_1, x_2, x_3\} = \{1,0,0,0\}$. Then the DFT of the discrete signals x_n is:

$$\begin{aligned} X_k &= \sum_{n=0}^{3} x_n \cdot e^{-\frac{2\pi ikn}{4}} \\ &= 1+0+0+0 = 1 \end{aligned}$$

Similarly, the DFT of the discrete signals $x_n = \{x_0, x_1, x_2, x_3\} = \{0,1,0,0\}$ is:

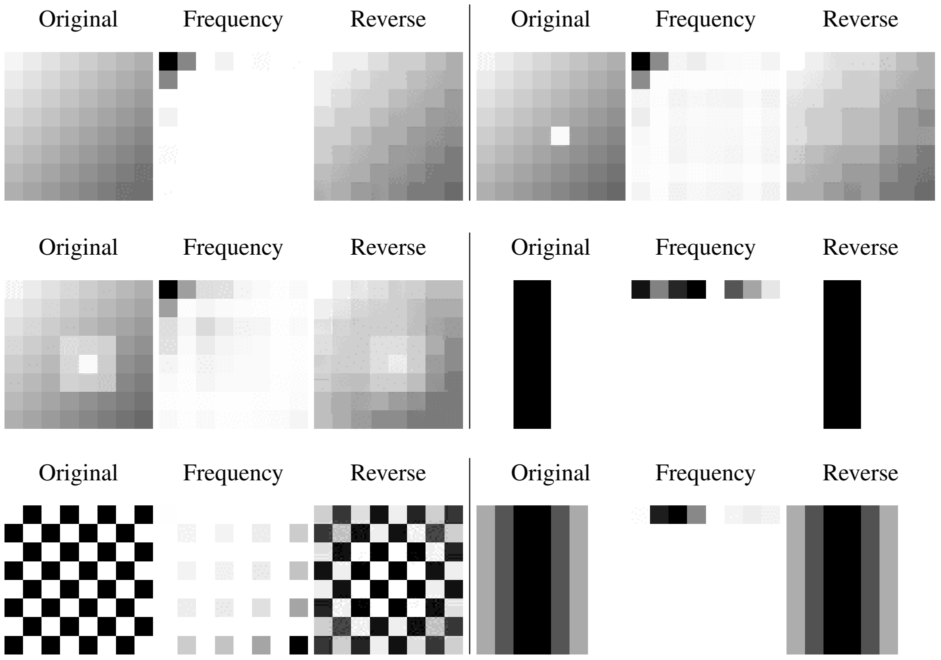


Figure 3.11: JPEG transformation example with original images (left), their corresponding frequency domain (middle), and reversed back to time domain images (right). [Picture credit: Dr. Mark Dunlop, The University of Strathclyde[1]]

$$\begin{aligned} X_k &= \sum_{n=0}^{3} x_n \cdot e^{-\frac{2\pi ikn}{4}} \\ &= e^{-\frac{2\pi ikn}{4}} \end{aligned}$$

So $X_0 = 1, X_1 = -i, X_2 = -1, X_3 = i$ or $X_k = \{1, -i, -1, i\}$ with $0 \leq k \leq N-1$.

Example 2: Figure 3.11 shows a classic example of the first step in a JPEG transformation, which is to convert the image into the frequency domain. In this figure, the original image on the left and a normalised version of the frequency domain image in the centre. The frequency domain images show the magnitude of each cell scaled from 0-255 with white representing 0 and 255 black. The reversed images from the frequency domain are on the right (after removing some high frequency information).

[1] https://personal.cis.strath.ac.uk/mark.dunlop/teaching/jpeg/

3.5.1 Fast Fourier Transform (FFT)

The time taken to evaluate a DFT on a digital computer depends principally on the number of multiplications involved, since these are the slowest operations. With the DFT, this number is directly related to order $O(N^2)$ operations (matrix multiplication of a vector), where N is the length of the transform. For most problems, N is chosen to be at least 256 in order to get a reasonable approximation for the spectrum of the sequence under consideration—hence computational speed becomes a major consideration.

The term Fast Fourier Transform (FFT) refers to a family of efficient algorithms for implementing DFT. FFT can significantly reduce the DFT computation of length N from order $O(N^2)$ to order $O(\frac{N}{2}\log_2 N)$ operations. This saving in computation is considerable when N is large. For example, for $N = 8192$, the number of operations for FFT is about 630 times fewer than a straightforward DFT. Here we present a particular algorithm to compute FFT called the Decimation-in-time algorithm. The principle of this approach is to divide a N-point DFT (N is a power of 2) into weighted sum of progressively shorted DFTs.

If a N-point signal is split into two parts, each with $\frac{N}{2}$ samples, one for elements at k odd- and the other for k even-ordered samples. Substituting $m = \frac{k}{2}$ for i even, and $m = \frac{k-1}{2}$ for i odd, we have:

$$F[n] = \sum_{m=0}^{\frac{N}{2}-2} f[2m]W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f[2m+1]W_N^{(2m+1)n}$$

As $\quad W_N^2 mn = e^{-j\frac{2\pi}{N}(2mn)} = e^{\frac{-j\frac{2\pi}{N}mn}{2}} = W_{\frac{N}{2}}^{mn}$, we have:

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m]W_{\frac{N}{2}}^{mn} + W_N^m \sum_{m=0}^{\frac{N}{2}-1} f[2m+1]W_{\frac{N}{2}}^{mn}$$

3.5.1.1 Windowing

As defined above, the Fast Fourier Transform (FFT) is the Fourier Transform of a block of time data points. The FFT transform assumes that this finite data set is a continuous spectrum that is one period of a periodic signal. When the signal is periodic and includes an integer number of periods, the result of the FFT turns out fine as it matches this assumption (see Figure 3.12 left and middle).

However, often the input signal is not an integer number of periods, and consequently results in discontinuous endpoints. These artificial discontinuities show up in the FFT as high-frequency components not present in the original signal. This phenomenon is known as spectral leakage, which causes the fine spectral lines to spread into wider signals (see Figure 3.12 right).

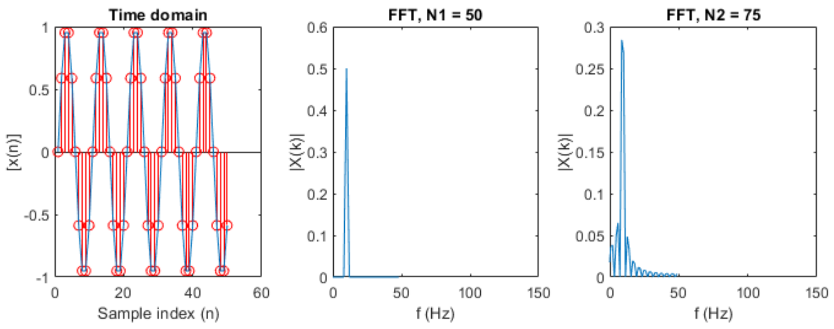


Figure 3.12: Illustration of spectral leakage: 0.5 seconds of 10Hz signal is sampled at 100Hz (left); FFT of a 50 data points window (middle) and 75 data points window of this signal (right).

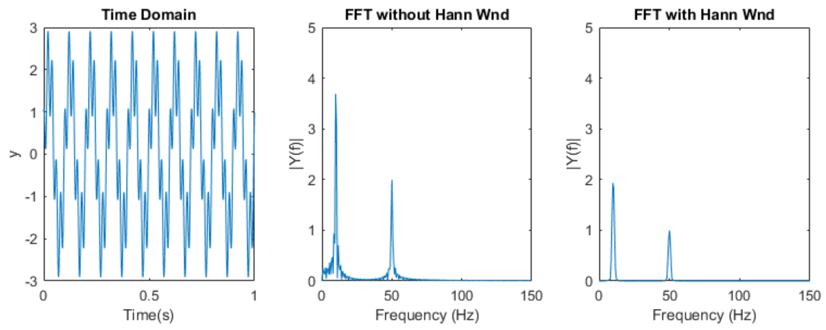


Figure 3.13: A time series of 10Hz and 50Hz signal (left), its FFT without using Hann window (middle) and with Hann window (right).

The Windowing technique can be used to minimize this effect of spectral leakage, by reducing the amplitude of data points at the edges to zero, so there is no discontinuity. This is done by multiplying the time record by a finite-length window with an amplitude that changes smoothly toward zero at the edges. Consequently, the endpoints of the waveform meet, resulting in a continuous waveform without sharp transitions. Figure 3.13 shows an example of applying FFT without Hann window (middle) of a 10Hz and 50Hz signal (left), and the FFT with Hann windowing (right).

There are many types of windowing functions, which should be used based on the signal type(s) and applications. Table 3.1 suggests an initial choice but it one should always compare the performance of different window functions to find the best one for the application.

Table 3.1: Window functions for different signal types.

Signal types	**Window function**
Sine wave or combination of sine waves	Hann
Sine wave (amplitude accuracy is important)	Flat Top
Narrowband random signal (vibration data)	Hann
Broadband random (white noise)	Uniform
Closely spaced sine waves	Uniform, Hamming
Excitation signals (hammer blow)	Force
Response signals	Exponential
Unknown content	Hann
Two tones with frequencies close but amplitudes very different	Kaiser-Bessel
Two tones with frequencies close and almost equal amplitudes	Hann
Accurate single tone amplitude measurements	Flat Top

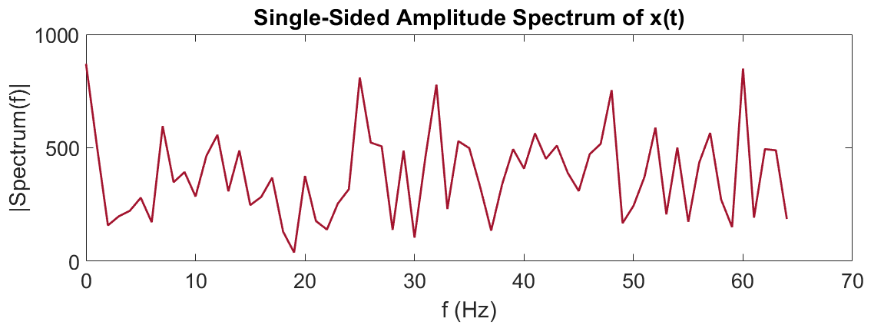


Figure 3.14: Single side amplitude spectrum of EEG signals x(t) (as described in Section 3.5.1.2).

3.5.1.2 Example

In Matlab, the Fast Fourier Transform can be computed using the function ***fft*** (Listing 3.1). Figure 3.14 illustrates the spectrum (single-sided) of the EEG signal: $X(t)$, which is a one-second length of EEG signal collected at the F3 channel (left frontal part of the brain [Rojas et al. 2018]) using an Emotiv™ EPOC neuroheadset [Emotiv 2019]. The signal was captured when participants closed their eyes and relaxed. The Matlab code used to produce Figure 3.14 is shown in Listing 3.2.

Listing 3.1: Fast Fourier Transform in Matlab with example

```
% input signal X: one-second length EEG signals
X = 1.0e+03 * ...
    [0.6662, -0.3918, 1.2031, 0.2528, 1.5631, -3.9097, 1.6257, ...
        -3.6410, 0.3277, -4.3323, -4.4010, -0.6620, -3.7892, 1.0969, ...
        1.9923, 1.1913, -3.5523, 0.4959, 2.1698, 2.9703, -1.3538, ...
        -3.4759, -3.6636, 2.3877, 0.9492, 1.2585, -4.6015, 0.9287, ...
        1.8390, 0.0631, -0.7369, -3.6856, -0.8072, 1.8518, 2.3708, ...
        -1.7661, 1.8200, 1.3205, 1.8790, -4.5108, 1.6385, -4.4543, ...
        -4.3061, -0.9595, 1.6277, -2.2082, -4.4682, 2.0318, 2.4036, ...
        -1.7974, 2.9390, 0.0375, -4.0031, 2.9954, -3.3108, -3.2077, ...
        3.0457, -2.8313, 0.1236, 0.4467, -1.0472, -2.9543, -3.6092, ...
        -1.1646, 0.8862, 0.5098, -1.8569, -2.0277, 1.9277, 0.6144, ...
        -2.4405, -1.8790, 1.2077, 0.2036, -3.1349, 2.0585, -2.5620, ...
        0.3154, -4.4461, -4.1241, 1.9826, 0.0764, -2.4774, 2.0175, ...
        -2.8825, 1.2662, -3.9590, -0.2020, -4.4554, 1.8816, -1.7502, ...
        1.0272, -2.5302, 0.4554, -2.0369, -2.6677, -1.5143, -4.6677, ...
        -1.6943, -4.5108, -0.9231, -0.1667, -1.2369, -0.1400, 1.2087, ...
        1.5759, -4.6415, -2.5790, -2.8831, 2.0595, -3.8215, -4.9123, ...
        2.9657, -2.6723, -4.7010, 0.9698, -3.9938, -1.3574, -1.6595, ...
        -1.5738, 2.5636, 2.1118, -2.3554, -0.6713, -1.7123, -1.9338, ...
        -1.2195, 1.7872];
% This function computes the discrete Fourier transform (DFT) of X ...
    using a fast Fourier transform (FFT) algorithm.
Y = fft(X);
% If X is a vector, then the output is FFT of the vector.
% If X is a matrix, then the output is FFT of each column of X.
```

Listing 3.2: Matlab code used to generate Figure 3.14

```
Fs = 128;              % Sampling frequency of Emotiv EPOC+
T = 1/Fs;              % Sampling period
L = 128;               % Length of signal
% Compute the Fourier transform of the signal.
y = fft(x);
% Compute the two-sided spectrum P2. Then compute the single-sided ...
    spectrum P1 based on P2 and the even-valued signal length L.
P2 = abs(y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
% Define the frequency domain f and plot the single-sided amplitude ...
    spectrum P1.
f = Fs*(0:(L/2))/L;
plot(f,P1)
title('Single-Sided Amplitude Spectrum of x(t)')
xlabel('f (Hz)')
ylabel('|Spectrum(f)|')
```

3.5.2 Digital Filtering

A digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. It is an important part of any DSP system. A digital filter is mainly used for two purposes:

- To separate a combined signal into different signals based on their sources.
- To restore signals that have been distorted in some way.

An example of the former is a measured brain activity (using EEG) that has been contaminated with interference (i.e. power frequency of 50Hz or 60Hz), noise (from nearby electronics), or other signals (i.e. eye movements and facial muscles). Conversely, signal restoration is used to estimate an original input signal that has been distorted with noise.

Digital filters can also be grouped into two types: finite impulse response (FIR) and infinite impulse response (IIR). The impulse response, often denoted as $h[k]$ or h_k, is a characterization of the filter's behaviour in response to the Kronecker delta function:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

In the case of FIR filters, the impulse response will decay to zero after a finite number of samples. In most cases, the impulse response is 'finite' because there is no feedback in the FIR. However, in some cases, it is still called a FIR filter if a feedback is used but the impulse response is finite. For example, a moving average filter is a FIR filter that has the N^{th} prior sample is subtracted (fed-back) each time a new sample appears. This filter has a finite impulse response even though it uses feedback: after N samples of an impulse, the output will always be zero.

This 'finite' number of samples is exactly equal to the sequence of filter coefficients:

$$y_n = \sum_{k=0}^{N} h_k x_{n-k}$$

A IIR filter, on the other hand, has an impulse response that is 'infinite' (never decays to zero). It is recursive and often designed based on a continuous-time transfer function. In other words, the impulse response is 'infinite' because there is a feedback in the filter. The output of a IIR filter depends on both current and previous inputs ($x[n-k]$ with $k \geq 0$) as well as previous outputs ($y[n-m]$ with $m > 0$):

$$\sum_{m=0}^{M} a_m y_{n-m} = \sum_{k=0}^{N} b_k x_{n-k}$$

If we rewrite the above equation to have $y[n]$ on the left and other terms on the right, then we have the output of the IIR filter $y[n]$ at time n:

$$y[n] = \sum_{k=0}^{N} b_k x_{n-k} - \sum_{m=1}^{M} a_m y_{n-m}$$

3.5.2.1 Comparisons of FIR and IIR filters

The key properties of FIR and IIR filters are:

- FIR filters can achieve linear phase response and pass a signal without phase distortion.
- FIR filters are often easier to implement than IIR filters.
- FIR filters sometimes require more memory (and calculation) to achieve a given filter response characteristic.
- FIR filters are better suited for applications that require a linear phase response (e.g. speech analysis and modelling, averaging, linear interpolation).
- IIR filters have non-linear phase response that distorts the phase of the input signals.
- IIR filters can achieve a given filtering characteristic using less memory and calculations than a similar FIR filter.
- IIR filters are suitable for applications that require no phase information (e.g. monitoring the signal amplitude).

3.5.2.2 Types of Digital Filters

There are three main types of digital filters:

High-pass and Low-pass: A digital high-pass filter passes signals with a frequency higher than a certain cutoff frequency and attenuates (suppresses or reduces the strength of) signals with frequencies lower than the cutoff frequency. Conversely, a digital low-pass filter passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. In both filters, the amount of attenuation for each frequency depends on the filter design.

Figure 3.15 illustrates the output (orange) of a high- (left) and low-pass filter of the same signal, sampled at 1kHz for 1 second (blue). The signal contains two tones, one at 100Hz and another at 200Hz. The high frequency tone has twice the amplitude of the low frequency tone. The high-pass filter of 150Hz removes the low-frequency tone (100Hz) and the low-pass filter removes the high-frequency tone (200Hz).

Band-pass: A high- and a low-pass filters can be used in conjunction to create a band-pass filter, which passes signals within a range of two cutoff frequencies and attenuates signals with frequencies outside that range.

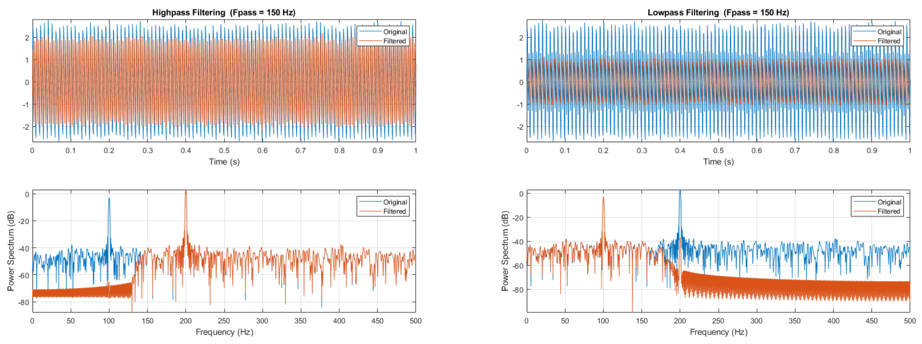


Figure 3.15: High-pass (left) and Low-pass (right) example of the same signal.

Listing 3.3: Matlab code used to generate Figure 3.16

```
Fs = 128;   % sampling frequency
% highpass with 4Hz cutoff frequency
y_highpass = highpass(x,4,Fs);
% lowpass with 8Hz cutoff frequency
y_lowpass = lowpass(x,8,Fs);
% bandpass with [4,8]Hz cutoff frequency
y_bandpass = bandpass(x,[4,8],Fs);
% plot all three output in one figure
subplot(3,1,1); plot(y_highpass);
subplot(3,1,2); plot(y_lowpass);
subplot(3,1,3); plot(y_bandpass);
```

Stop-band (Notch): A Notch filter is the complement of a band-pass filter. It attenuates signals with frequencies within a certain narrow band of frequency, and passes all other signals with frequencies outside that narrow band of frequency.

Figure 3.16 illustrates these three filters using the sample EEG signal described in Section 3.5.1.2 as their input. The cutoff frequencies are 4Hz and 8Hz. These frequencies are used to detect a specific EEG pattern called Error Related Negativity, observed after errors are committed during various choice tasks [Vi et al. 2014b]. The corresponding Matlab code is shown in Listing 3.3.

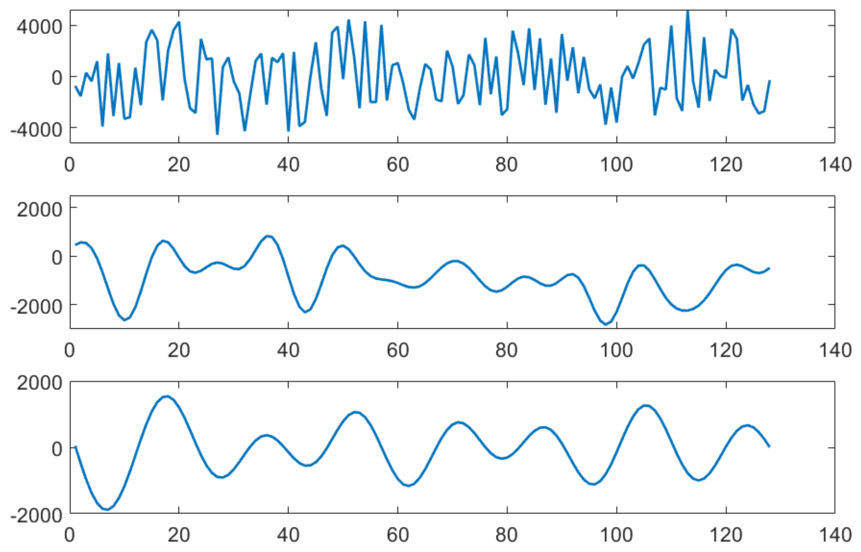


Figure 3.16: Highpass (top), lowpass (middle), and bandpass (bottom) of sample EEG signals described in Section 3.5.1.2.

3.5.2.3 Filter Order

The order of a filter is the maximum delay, in number of samples, that is used to create the output [Smith 2007]. They are the values of N (for FIR filter) and M (for IIR filter) in the above equations. With this definition, an example of a second-order FIR filter is:

$$\begin{aligned} y_n &= \sum_{k=0}^{2} h_k x_{n-k} \\ &= x_n - x_{n-1} + 2x_{n-2} \end{aligned}$$

Similarly, an example of a second-order IIR filter is:

$$\begin{aligned} y_n &= \sum_{k=0}^{1} b_k x_{n-k} - \sum_{m=1}^{2} a_m y_{n-m} \\ &= x_n - x_{n-1} - 3y_{n-1} + 2y_{n-2} \end{aligned}$$

3.6 Autocorrelation

The correlation of a signal with a delayed copy of itself is its autocorrelation. This often occurs in a time series where each observation (a data point) is somewhat dependent upon the previous observation (called 'lag'). In other words, it is the same as calculating the correlation between two different time series, except that the same time series is actually used twice: once in its original form and once lagged one or more time periods.

Why is it important? As autocorrelation measures the similarity between observations (samples) as a function of the time lag between them, it can be used to identify recurring data patterns and thus detect non-randomness in the observations (and the data that they belong to).

3.6.1 The correlation coefficient

The first step is to find a way of measuring how similar two time series are. The most used method for this purpose is called *correlation*. The correlation between two time series is a measure of how similarly they behave. It can be expressed as:

$$corr(X,Y) = \frac{cov(X,Y)}{std(X)std(Y)}$$

with *mean(X)* is the average of all data points in the time series:

$$std(X) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[X_i - mean(X)]^2}$$

And the *std(X)* is an indication of how much the series is distant themselves from the mean.

$$mean(X) = \frac{1}{N}\sum_{i=1}^{N} X_i$$

The above two terms are often associated with *variance*:

$$var(X) = std(X)^2$$

The term $cov(X,Y)$ represents the covariance between X and Y, which generalises the concept of variance over two time series instead of one. The covariance provides a measure of how much two time series change together. It does not necessarily account for how similar they are, but for how similarly they behave. More in detail, it captures whether the two time series increase and decrease at the same time.

The covariance is calculated as follow:

$$cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}[X_i - mean(X)][Y_i - mean(Y)]$$

3.6.2 Autocorrelation Function

Once we understand the concept of correlation, the concept of autocorrelation can be understood as the correlation coefficient of a time series with itself, shifted in time. If the data has a periodicity, the correlation coefficient will be higher when those two periods resonate with each other.

The first step is to define an operator to shift a time series in time, causing a delay of t (also known as *lag*):

$$lag(X_i, t) = X_{i-t}$$

The autocorrelation of a time series with lag t is defined as:

$$autocorr(X,t) = corr[X, lag(X,t)]$$

which can also be written as:

$$autocorr(X,t) = \frac{cov[X, lag(X,t)]}{std[X]std[lag(X,t)} = \frac{cov[X, lag(X,t)]}{var(X)}$$

or

$$autocorr(X,t) = \frac{\sum_{i=1}^{N}[X_i - mean(X)][X_{i-t} - mean(X)}{\sum_{i=1}^{N}[X_i - mean(X)]^2}$$

3.6.3 Application of Autocorrelation

Autocorrelation has been used widely in various interactive applications in HCI, especially to process different types of physiological signals. Some examples include using short-time autocorrelation technique on electrocardiogram signals (ECG) to detect instantaneous heart rate [Nakano et al. 2012], to detect movement intention [Wairagkar et al. 2018] and classification of different mental tasks using EEG signals [Rahman et al. 2018], or to investigate the impact of differences in meal ingestion amount on electrogastrograms (electrical signals that travel through the stomach muscles and control the muscles' contractions) [Kinoshita et al. 2019]. Further applications of autocorrelation in signal processing are listed below:

- In signal processing, autocorrelation can provide information about repeating events such as musical beats or pulsar frequencies, but it cannot tell the position in time of the beat. It can also be used to estimate the pitch of a musical tone.
- In music recording, autocorrelation is used as a pitch detection algorithm prior to vocal processing, as a distortion effect or to eliminate undesired mistakes and inaccuracies.
- In statistics, spatial autocorrelation between sample locations also helps one estimate mean value uncertainties when sampling a heterogeneous population.

- In astrophysics, auto-correlation is used to study and characterize the spatial distribution of galaxies in the Universe and in multi-wavelength observations of Low Mass X-ray Binaries.
- Autocorrelation analysis is widely used in fluorescence correlation spectroscopy (i.e., to visualize blood flow in medical ultrasound imaging).
- Autocorrelation is used to measure the optical spectra and to measure the very-short-duration light pulses produced by lasers.
- Autocorrelation is used to analyze dynamic light scattering data for the determination of the particle size distributions of nanometer-sized particles in a fluid. A laser shining into the mixture produces a speckle pattern. Autocorrelation of the signal can be analyzed in terms of the diffusion of the particles. From this, knowing the fluid viscosity, the sizes of the particles can be calculated using Autocorrelation.
- The small-angle X-ray scattering intensity of a nano-structured system is the Fourier transform of the spatial autocorrelation function of the electron density.
- In optics, normalized autocorrelations and cross-correlations give the degree of coherence of an electromagnetic field.
- In an analysis of Markov chain Monte Carlo data, autocorrelation must be taken into account for correct error determination.

3.7 Linear Time-Invariant Systems

Linear time-invariant (LTI) systems are systems that are both linear and time-invariant. A linear system takes a linear combination of inputs, and outputs a linear combination of individual responses to those inputs (i.e. Figure 3.17). A time-invariant system always gives the same output that does not depend on when an input was applied (i.e. Figure 3.18). These two properties make LTI systems easy to understand and represent graphically.

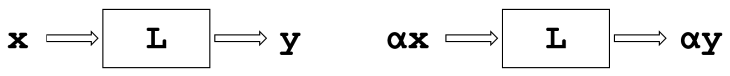


Figure 3.17: In this example, an input *x* to the linear system *L* gives the output *y* (left). If x is scaled by a value α, output y of the same system will also be scaled by α (right).

3.7.1 Convolution

Before exploring the properties of a LTI system, we need to understand Convolution. Convolution is a mathematical approach for combining two signals to form a third signal. Using the strategy of impulse decomposition, which breaks an N samples signal into N component

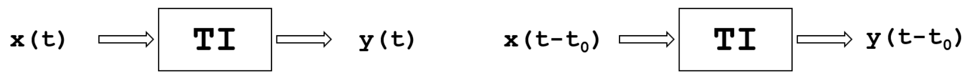


Figure 3.18: In this example, an input x at time t passes through a time-invariant system will have an output y (left). If the input is delayed t_0 seconds later, the output is the same except it is linked to the delayed t_0 (right).

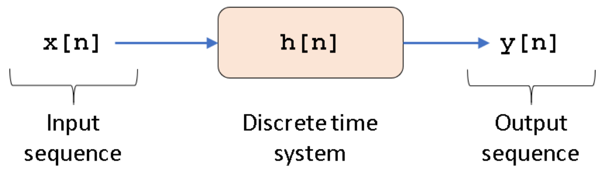


Figure 3.19: Illustration of the convolution of the input signal ($x[n]$) with the impulse response of the system ($h[n]$) to produce the output ($y[n]$).

signals—each contains one point from the original signal with the remainder of the values being zero, systems are described by signals called the impulse responses. Convolution is important because it relates the three signals of interest: the input signal ($x[n]$), the output signal ($y[n]$), and the impulse response ($h[n]$) as illustrated in Figure 3.19.

Mathematically, if we define $y[n]$ as the convolution of the (input) signal $x[n]$ and the (impulse response) signal $h[n]$, then we can write:

$$y[n] = x[n] * h[n] = \sum_{m=-\infty}^{\infty} x[m] \cdot h[n-m]$$

Figure 3.20 illustrates the convolution of $x = [0,0,0,1,1,1,0,0,0,0]$ with $h = [0.5, 0.5, 0.5, 0,0,0,0,0,0,0]$ to get $y = x * h = [0,0,0.5,1.0,1.5,1.0,0.5,0,0,0,0,0,0,0,0,0,0,0,0]$.

3.7.2 Properties of a LTI system

A linear time-invariant system has the following fundamental properties:

Identity Property

$$x[n] * \delta[n] = x[n]$$

Here, $\delta[n]$ is the neutral element of the convolution operation. The implication of this property is that we can express time-shifting as a convolution with a time-shifted δ:

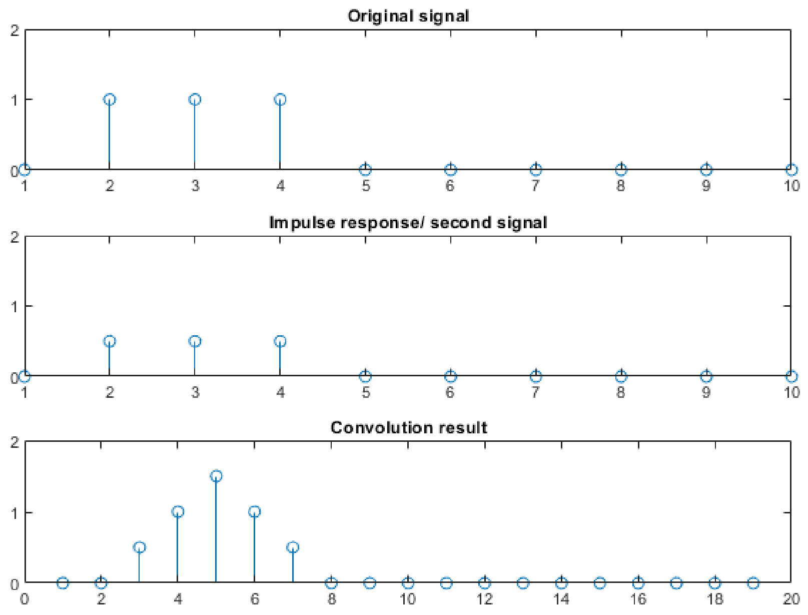


Figure 3.20: Illustration of the convolution result (bottom) of an input signal (top) with an impulse response signal (middle).

$$x[n] * \delta[n - t_0] = x[n - t_0]$$

Commutative Property

$$x[n] * h[n] = h[n] * x[n]$$

The commutative property means simply that the input signal x convolved with the impulse response h is identical with h convolved with x. The consequence of this property for LTI systems is that the output will be the same if the roles of the input and impulse response are interchanged.

Distributive Property

$$x[n] * (\alpha h_1[n] + \beta h_2[n]) = \alpha(x[n] * h_1[n]) + \beta(x[n] * h_2[n])$$

where α, β are constants.

The distributive property shows that the convolution of a signal with the sum of two other signals is identical to the sum of the convolution with each signal in the sum individually and then summing the two results. This means in LTI systems with parallel combination, the output can be changed into a single system whose impulse response is the sum of the two individual ones.

Associative Property

$$x[n] * (v[n] * w[n]) = (x[n] * v[n]) * w[n]$$

The associative property indicates that the convolution of three signals is ambiguously without concerning about how they are grouped pairwise.

3.8 Use Case Example

In this section, we illustrate how DSP is used in processing Electroencephalography (EEG) signals. EEG is a sensing technology that uses electrodes placed on the scalp to measure electrical potentials related to brain activity. The EEG device records the voltage at each of these electrodes relative to a reference point, which is often another electrode on the scalp. As a result, the EEG signals are time continuous signals and need to go through all processing steps before they can be analyzed. Figure 3.21 illustrates this process.

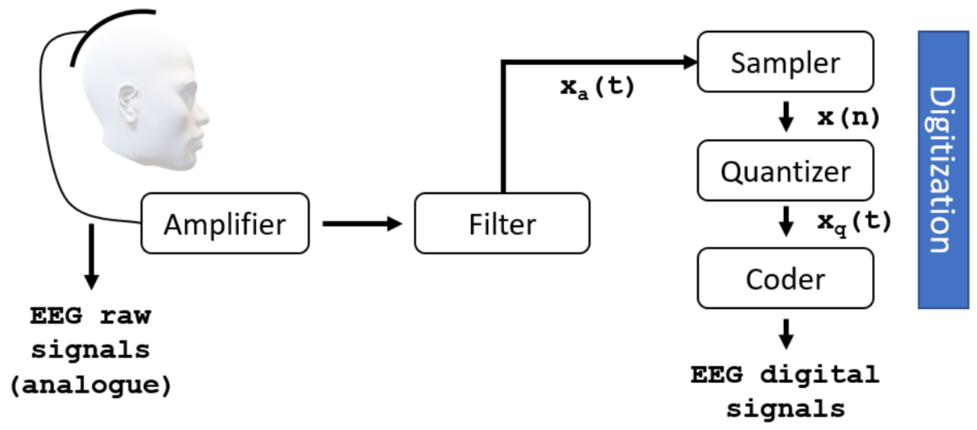


Figure 3.21: Illustrative steps of processing Electroencephalography (EEG) signals from their analogue to digital forms.

The captured raw EEG signals often has amplitude in microvolts and frequency up to 300Hz. They first need to be amplified and filtered either before or after the analog-to-digital conversion (ADC), to reduce noise and make the signals suitable for digitally processing later. The filtering step is mainly to remove contaminated noise such as breathing (need a high-pass filter), high-frequency noise (i.e. >50Hz that needs a low pass filter), or the power supply frequency (i.e. 50Hz that needs notch filter).

After this, EEG signals are converted into digital form. This digitization process contains steps of sampling, quantization, and encoding (see Figure 3.21). These steps are often carried out by the software capturing the EEG signals. From here, EEG signals can be further processed to understand the underlying neural processes. For example, the digitally filtered EEG signals can be interpreted based on the frequency bands. More details of processing EEG signals are described in Chapter 6 (Brain-Computer Interfacing with Interactive Systems).

- Delta (<4 Hz): This frequency is dominant in EEG of up to one-year infants. It also appears in the last two stages of sleep (NREM stage 3 and REM).
- Theta (4 to 7 Hz): This band is dominant in EEG of users with drowsiness. Theta waves may also appear during hypnagogic states such as hypnosis, deep daydreams, and the state just before falling asleep (preconscious stage).
- Alpha (8 to 13 Hz): This band has strong appearance when users are relaxed. It is also connected to variations in arousal or sleep. Usually the eyes close leads to the rise of alpha rhythms. Additionally, alpha frequencies reduce when users are drowsy and open eyes. A variant of the alpha band named Mu-rhythm is occasionally captured in the motor cortex and reduces its amplitude with the intention to make a movement.
- Beta (14 to 30 Hz). This band is considered as an index of cortical arousal. Beta frequencies with low amplitude are usually connected with active or nervous thoughts and active concentration.

The above frequency bands can be computed to correspond to user's cognitive states. For example, it has been demonstrated that user's engagement in an interactive task corresonds to *beta/(alpha + theta)*. This meaningful metric has been used to compare and contrast interactive tasks [Vi et al. 2013], an input method [Vi et al. 2014a] in games, or an indicator for switching the mode of operation of a task set (e.g., manual/automated) in a biocybernetic system [Pope et al. 1995]. Further examples of using EEG to detect cognitive load in driving situations are described in Chapter 11 (Driver Cognitive Load Classification based on Physiological Data).

3.9 Follow-up Questions

To practice some topics covered in this chapter, we will use an example of an EEG-based brain computer interface from Guger et al. [2009]. First, you need to download an example dataset, in Matlab format from https://lampx.tugraz.at/~bci/database/003-2015/s1.mat. The data was recorded using an 8-channel EEG device, with a more detailed description from https://lampx.tugraz.at/~bci/database/003-2015/description.pdf. Once imported into Matlab, practice the following tasks:

- Select one EEG channel among the 8 EEG channels of the train set.
- Segment the this channel data into 2s windows, with 50% overlaps between windows.
- Filter the data using a bandpass filter of [0.5 Hz – 30 Hz].
- Filter the data into different frequency bands, as described in section 3.8.
- Visualize the differences between different frequencies on a sub-plot.

3.10 Summary

This chapter has provided an introduction to Digital Signal Processing for Interactive Systems designers. It demonstrated the diverse set of signal sources and the importance of a basic understanding of signal processing for inventors of systems that interact with the physical world. It covered signal classification, analog-to-digital conversion, digital-to-analog conversion, the Discrete Fourier Transform, autocorrelation, linear time-invariant systems and the use of DSP in an example application—processing Electroencephalography (EEG) signals. A reader seeking a deeper understanding should consult the texts outlined in the introduction of the chapter. We hope this introduction provides the reader with the necessary background to begin development of interactive systems that utilise Digital Signal Processing.

Bibliography

Arduino AG, 2018a. Arduino – Home. https://www.arduino.cc/. Last accessed: 15/08/2018.

Arduino AG, 2018b. Arduino Uno Rev3. https://store.arduino.cc/arduino-uno-rev3. Last accessed: 15/08/2018.

J. Cassell, J. Sullivan, E. Churchill, and S. Prevost. 2000. *Embodied Conversational Agents*. MIT press.

Emotiv, 2019. Emotiv EPOC+ Neuroheadset. https://www.emotiv.com/epoc/. Last accessed: 26/04/2019.

T. Grosse-Puppendahl, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, S. Hodges, M. S. Reynolds, and J. R. Smith. 2017. Finding Common Ground: A Survey of Capacitive Sensing in Human-Computer Interaction. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pp. 3293–3315. ACM, New York, NY, USA. ISBN 978-1-4503-4655-9. DOI: 10.1145/3025453.3025808.

C. Guger, S. Daban, E. Sellers, C. Holzner, G. Krausz, R. Carabalona, F. Gramatica, and G. Edlinger. 2009. How many people are able to control a p300-based brainâ€"computer interface (bci)? *Neuroscience Letters*, 462(1): 94 – 98. ISSN 0304-3940. http://www.sciencedirect.com/science/article/pii/S0304394009008192. DOI: https://doi.org/10.1016/j.neulet.2009.06.045.

E. Jones, J. Alexander, A. Andreou, P. Irani, and S. Subramanian. 2010. GesText: Accelerometer-based Gestural Text-entry Systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pp. 2173–2182. ACM, New York, NY, USA. ISBN 978-1-60558-929-9. DOI: 10.1145/1753326.1753655.

C. Kiefer, N. Collins, and G. Fitzpatrick. 2008. HCI Methodology For Evaluating Musical Controllers: A Case Study. In *NIME*, pp. 87–90.

F. Kinoshita, K. Miyanaga, K. Fujita, and H. Touyama. 2019. *Effect of Differences in the Meal Ingestion Amount on the Electrogastrogram Using Non-linear Analysis*, pp. 468–476. ISBN 978-3-030-23562-8. DOI: 10.1007/978-3-030-23563-5_37.

P. Klasnja, S. Consolvo, and W. Pratt. 2011. How to Evaluate Technologies for Health Behavior Change in HCI Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pp. 3063–3072. ACM, New York, NY, USA. ISBN 978-1-4503-0228-9. DOI: 10.1145/1978942.1979396.

Micro:bit Educational Foundation, 2018. Micro:bit Educational Foundation — micro:bit. http://microbit.org/. Last accessed: 15/08/2018.

Microchip Technology Inc., 2018. ATmega328/P AVR® Microcontroller with picoPower® Technology. https://www.microchip.com/wwwproducts/en/ATmega328P. Last accessed: 15/08/2018.

M. Nakano, T. Konishi, S. Izumi, H. Kawaguchi, and M. Yoshimoto. 2012. Instantaneous Heart Rate Detection using Short-time Autocorrelation for Wearable Healthcare Systems. In *2012 Annual*

International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 6703–6706. DOI: 10.1109/EMBC.2012.6347532.

Nordic Semiconductor, 2018. nRF51822 Multiprotocol Bluetooth® Low Energy/2.4 GHz RF System on Chip Product Specification v3.3. https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822. Last accessed: 15/08/2018.

K. Pfeuffer, J. Alexander, M. K. Chong, Y. Zhang, and H. Gellersen. 2015. Gaze-Shifting: Direct-Indirect Input with Pen and Touch Modulated by Gaze. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pp. 373–383. ACM, New York, NY, USA. ISBN 978-1-4503-3779-3. DOI: 10.1145/2807442.2807460.

A. T. Pope, E. H. Bogart, and D. S. Bartolome. 1995. Biocybernetic system evaluates indices of operator engagement in automated task. *Biological Psychology*, 40(1): 187 – 195. ISSN 0301-0511. http://www.sciencedirect.com/science/article/pii/0301051195051163. DOI: https://doi.org/10.1016/0301-0511(95)05116-3. EEG in Basic and Applied Settings.

J. G. Proakis and D. G. Manolakis. 2007. *Digital Signal Processing*, 4. Pearson. ISBN 01318773741.

M. M. Rahman, M. A. Chowdhury, and S. A. Fattah. 2018. An Efficient Scheme for Mental Task Classification Utilizing Reflection Coefficients Obtained from Autocorrelation Function of EEG Signal. *Brain Inform*, 5(1): 1–12.

Raspberry Pi Foundation, 2018. Raspberry Pi – Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/. Last accessed: 15/08/2018.

T. K. Rawat. 2015. *Digital Signal Processing*. Oxford University Press. ISBN 9781680158731.

T. Rodden, K. Cheverst, K. Davies, and A. Dix. 1998. Exploiting Context in HCI Design for Mobile Systems. In *Workshop on Human Computer Interaction with Mobile Devices*, pp. 21–22.

G. Rojas, C. Alvarez, C. Montoya Moya, M. de la Iglesia Vaya, J. Cisternas, and M. Gálvez. 2018. Study of Resting-State Functional Connectivity Networks Using EEG Electrodes Position As Seed. *Frontiers in Neuroscience*, 12. DOI: 10.3389/fnins.2018.00235.

A. Schmidt. Dec. 2015. Biosignals in Human-computer Interaction. *Interactions*, 23(1): 76–79. ISSN 1072-5520. DOI: 10.1145/2851072.

H. P. D. Silva, S. Fairclough, A. Holzinger, R. Jacob, and D. Tan. Jan. 2015. Introduction to the Special Issue on Physiological Computing for Human-Computer Interaction. *ACM Trans. Comput.-Hum. Interact.*, 21(6): 29:1–29:4. ISSN 1073-0516. DOI: 10.1145/2688203.

J. Smith. 2007. *Introduction to Digital Filters: With Audio Applications*. Music Signal Processing Series. W3K. ISBN 9780974560717.

S. Smith. 2002. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing.

J. Suarez and R. R. Murphy. 2012. Hand Gesture Recognition with Depth Images: A Review. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pp. 411–417. DOI: 10.1109/ROMAN.2012.6343787.

F. Taher, J. Hardy, A. Karnik, C. Weichel, Y. Jansen, K. Hornbæk, and J. Alexander. 2015. Exploring Interactions with Physically Dynamic Bar Charts. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 3237–3246. ACM, New York, NY, USA. ISBN 978-1-4503-3145-6. DOI: 10.1145/2702123.2702604.

C. T. Vi, K. Takashima, H. Yokoyama, G. Liu, Y. Itoh, S. Subramanian, and Y. Kitamura. 2013. D-FLIP: Dynamic and Flexible Interactive PhotoShow. In D. Reidsma, H. Katayose, and A. Nijholt, eds., *Advances in Computer Entertainment*, pp. 415–427. Springer International Publishing, Cham. ISBN 978-3-319-03161-3.

C. T. Vi, J. Alexander, P. Irani, B. Babaee, and S. Subramanian. 2014a. Quantifying EEG Measured Task Engagement for use in Gaming Applications.

C. T. Vi, I. Jamil, D. Coyle, and S. Subramanian. 2014b. Error Related Negativity in Observing Interactive Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pp. 3787–3796. ACM, New York, NY, USA. ISBN 978-1-4503-2473-1. DOI: 10.1145/2556288.2557015.

M. Wairagkar, Y. Hayashi, and S. J. Nasuto. 2018. Exploration of Neural Correlates of Movement Intention based on Characterisation of Temporal Dependencies in Electroencephalography. *PLOS ONE*, 13(3): 1–23. https://doi.org/10.1371/journal.pone.0193722. DOI: 10.1371/journal.pone.0193722.

J. S. Wilson. 2004. *Sensor Technology Handbook*. Elsevier.

T. G. Zimmerman, J. R. Smith, J. A. Paradiso, D. Allport, and N. Gershenfeld. 1995. Applying Electric Field Sensing to Human-computer Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pp. 280–287. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0-201-84705-1. DOI: 10.1145/223904.223940.