Chapter 5

Signal and Data Processing

The Sensor-node Level

A smart sensor node should be able to operate autonomously and independently of the environment and other sensor nodes. It consists of different modules and units:

1. Digital Signal Processing and Data Processing Units (DSP/DPU: microprocessor, virtual machine, and digital logic and Register-Transfer Level architectures)
2. Communication Units (CMU: physical, link, and data level)
3. Digital Storage (DS)
4. Energy Management Units (EM: hardware and software control)
5. Analog Signal Processing (ASP: analog electronics)
6. Analog-Digital Conversion (ADC)
7. Energy Supply and Energy Storage (ES)

The relation of the single units and the data flow is illustrated schematically in Fig. **5.1**. Typically a sensor node processes low-level sensor data, improving the quality and accuracy using noise and fusion filter techniques. The sensor data or a pre-processed information is passed from this information source to an information sink node by using message passing routed in some kind of communication network.

 ISBN 978-3-527-33606-7

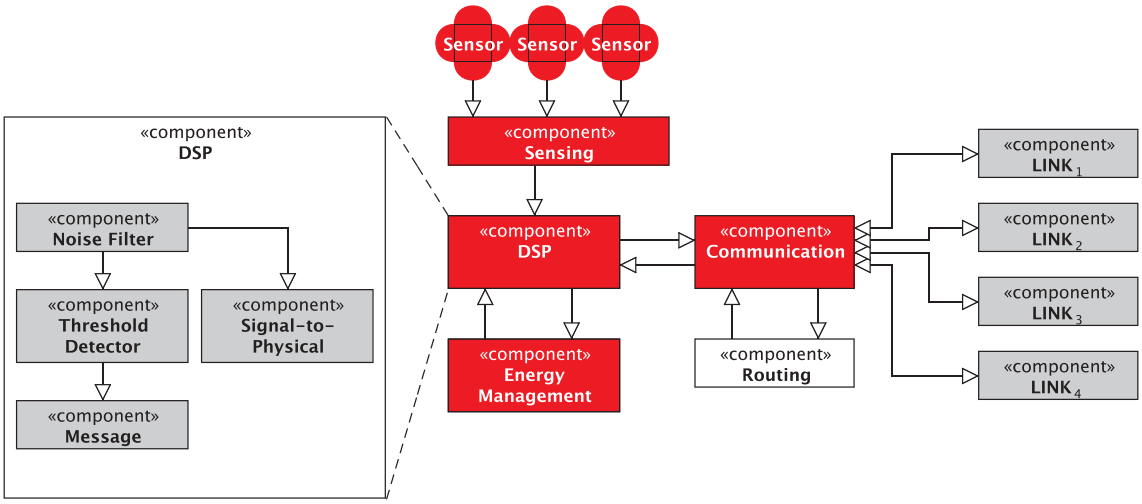


Fig. 5.1 *Architecture of a Smart Sensor Node*

This chapter outlines the principles of different components of a sensor node and the challenges to integrate sensor nodes in materials and mechanical structures.

5.1 Analog Sensor Signal Processing and Analog-to-Digital Conversion [Horstmann, Bosse]

Data acquisition systems building the interface between the physical parameter of the real world, which are analog and which are sensed by sensors, and the world of digital computation **[WEB14]**. The devices that perform the interface function between analog and digital worlds are analog-to-digital (ADC). ADC are key components in todays modern applications and are essential parts of sensing systems. In data acquisition systems the analog signal from the sensor has to be adjusted to suit the sampling frequency and the amplitude range needed by the analog-to-digital converter. The two main functions of an ADC are sampling and quantization. These two are converting the analog sensor signal from the continuous time and voltage into digital numbers having discrete values at discrete times. To convert analog signals continuously in time and at every voltage would take an infinite amount of storage. Therefore every data acquisition system has to be designed for an appropriate sampling rate and quantization, better described as resolution. In Figure **5.2** a functional overview of a data acquisition system is given. The sensor output is first amplified to suit the input range of the following ADC. Furthermore this functional block is used to convert the sensor output into voltages. The signal is then filtered by an anti-aliasing filter to remove high-frequency components that may cause an

ISBN 978-3-527-33606-7

effect known as aliasing. The signal is sampled and held by a circuit and converted into a digital representation.

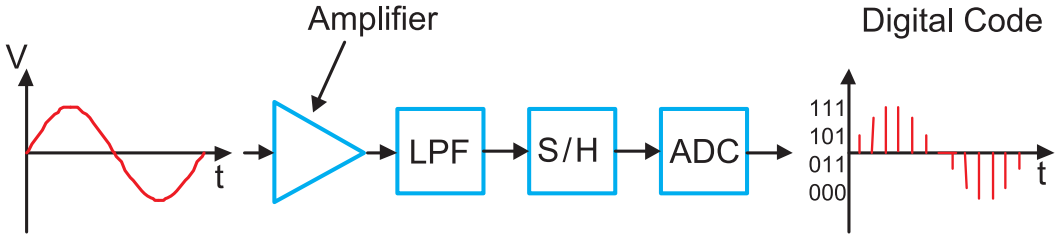


Fig. 5.2 *Signal characteristics and functional overview*

A deeper and rigorous introduction in measuring systems, their properties, and the design of measuring systems can be found in **[WEB14]** and **[SYD05]**.

5.1.1 Operational Amplifiers

The front end of the data acquisition system amplifies and filters the signal of the sensor node. An amplifier and filter are crucial components in the initial signal processing. The amplifier must perform at least one of the following functions: amplifying the signal, buffer the signal, convert a signal current into a voltage or extract a differential voltage from a common-mode noise **[JOH97]**. To accomplish these functions the most popular amplifier will be used. It is called operational amplifier (OPAMP) which is a general purpose gain block having differential inputs. By connecting the OPAMP with different closed-loop configurations a wide variety of functionalities can be realized. A selection of configurations is shown in Figure **5.3**.

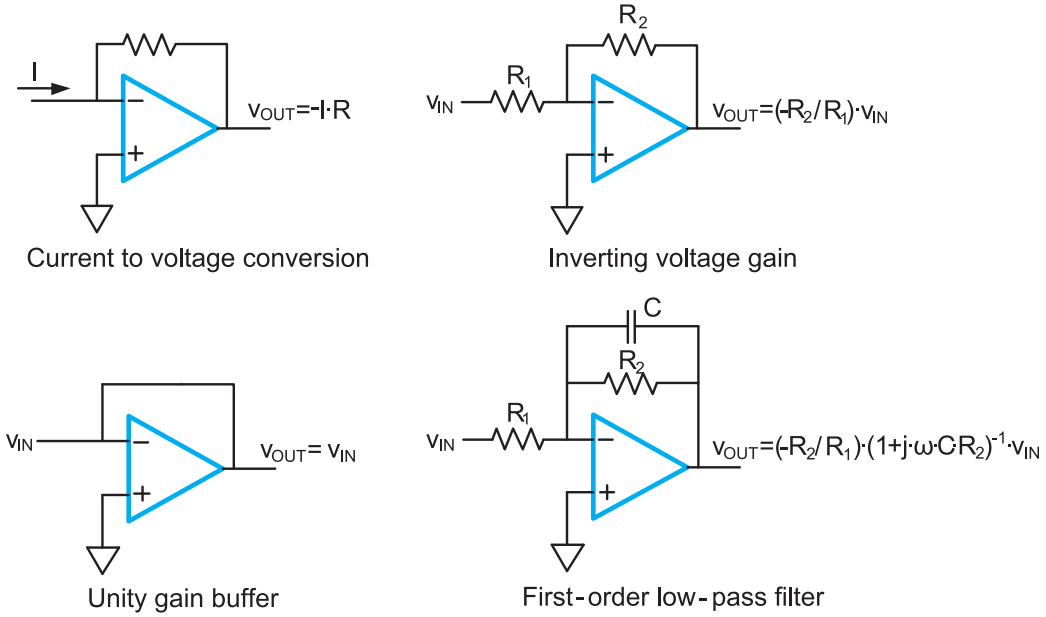


Fig. 5.3 *Operational amplifier circuits*

The gain of the circuits depends on the external devices around the amplifier. In case of differential signal processing the instrumentation amplifier is a better choice since both inputs having high impedances and the gain is set whit one precision resistor as shown in Figure **5.4**.

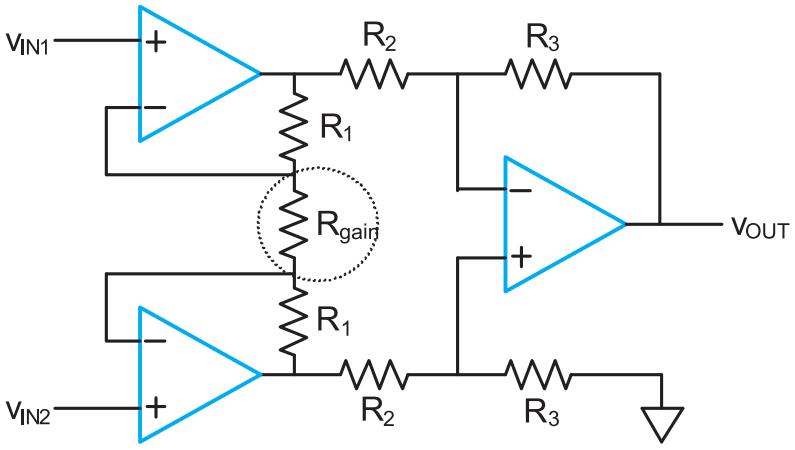


Fig. 5.4 *Instrumentation amplifier for high input impedance*

ISBN 978-3-527-33606-7

Each input is directly connected to an OPAMP working as a unity gain buffer if R_{gain} is left out. Since the inputs are internally connected to a differential stage, meaning a gate of a MOSFET, the impedance is very high. The rightmost amplifier is working as a standard differential amplifier circuit with a gain equal to R_3/R_2. By including R_{gain} the transfer function is to be described as

$$\frac{v_{out}}{v_{in2} - v_{in1}} = \left(1 + \frac{2R_1}{R_{gain}}\right)\frac{R_3}{R_2} \tag{5.1}$$

A further advantage of using an instrumentation amplifier is its high common mode rejection ratio (CMRR). An ideal differential input amplifier responds only to the voltage difference between its input terminals. An increase or decrease of the common-mode voltage at the input terminal should not change the output voltage. In non-ideal amplifiers the common-mode input signal causes an output respond even so small compared to the response of the differential input voltage. The common-mode rejection ratio is the ratio of differential voltage gain to common-mode voltage gain and can be described as

$$CMRR = 20\log\left(\frac{A_D}{A_{CM}}\right) \tag{5.2}$$

Where A_D is the differential voltage gain and A_{CM} the common-mode gain. Besides amplifying voltages the frequency behavior in the front-end data acquisition system is important. Figure **5.2** shows a low-pass filter to prevent an effect called aliasing and removing noise components at higher frequencies. The effect of inadequate sampling rate on a sinusoid is illustrated in Figure **5.5**.

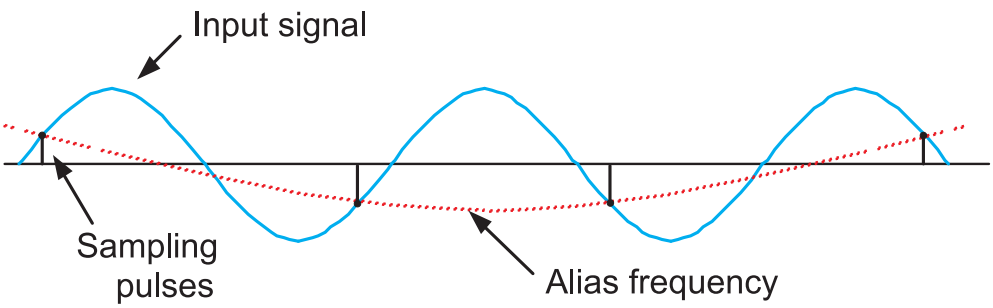


Fig. 5.5 *Alias frequency caused by under-sampling*

ISBN 978-3-527-33606-7

In this case, sampling at a rate slightly less than twice per cycle gives a low-frequency sinusoid shown as the dotted waveform. This alias waveform can be significantly different from the original frequency. In this figure it is easy to see, that if the sinus waveform is sampled at least twice per cycle, as required by the sampling theorem, the original frequency is preserved.

5.1.2 Analog-to-digital converter specifications

Sampling

The task of the sample-and-hold (S/H) circuit is to lock the value of an input signal at a certain time and to hold it while the ADC is processing the data. Figure **5.6** shows the behavior of an ideal S/H circuit and a simplified circuit. The input-signal is periodically captured and held at a constant voltage level until the next time step. As the performance of the S/H circuit it is essential to the quality of the data conversion, it is important to determine its characteristics in detail (see **[KES05]** for technical aspects).

Most S/H circuits employ operational amplifiers and thus non-ideal op-amp behavior has to be taken into account (see Figure **5.7**). If the input voltage is changing rapidly then the output of the amplifier could be limited by its slew-rate. Further issues may occur if the amplifier is not correctly compensated, so that the phase margin at the transit frequency is too small. This would result in a large overshoot and a longer settling time would be required until the output of the S/H circuit approaches the signal value within the necessary tolerances. Consequently, the circuit would require a larger acquisition time, i.e. the time interval between sample command and subsequent hold command. Additionally, the op-amps DC error should be considered. The overall output error depends on the amplifiers offset, linearity, and gain error. Ideally the gain of a S/H circuit is 1 and constant over the input voltage range.

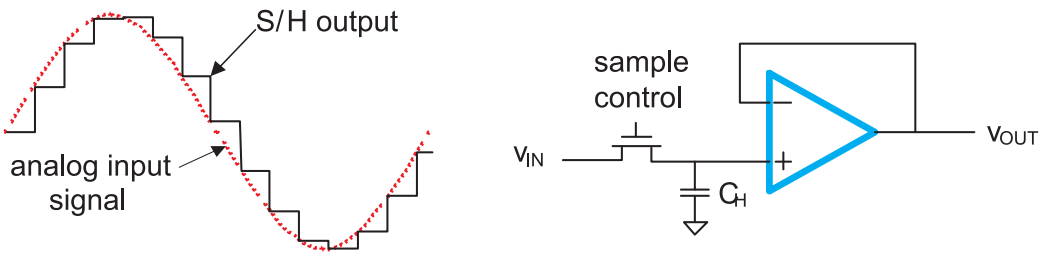


Fig. 5.6 *left) Output of an ideal S/H Circuit, right) Track-and-hold circuit using a buffer*

ISBN 978-3-527-33606-7

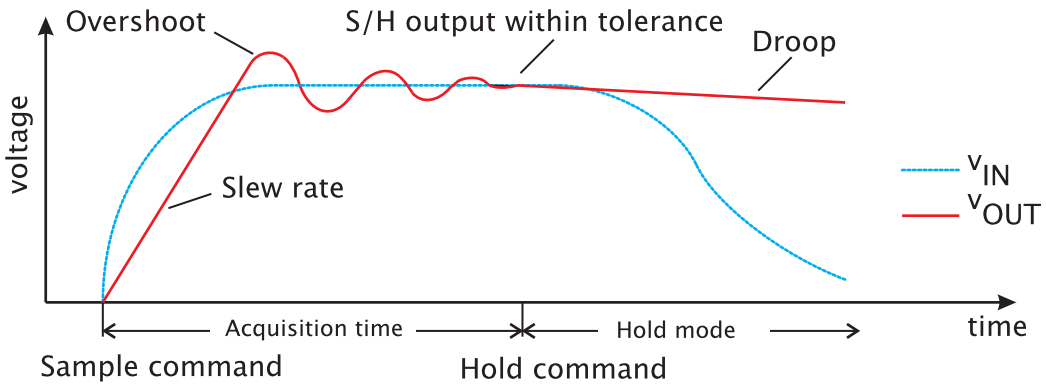


Fig. 5.7 *Typical errors caused by a sample-and-hold circuit*

Once the S/H circuit is in hold mode further deviations from the ideal behavior of the S/H circuit may arise. The parasitic impedances associated the following ADC as well as the amplifier of the S/H circuit may cause leakage currents, which in turn may lead to a drop of the voltage level. A larger hold capacitor can mitigate this effect. However, this will result in a longer acquisition time, as the time needed to charge the capacitor increases.

A transient effect which is inducing an error is called aperture error. Since the transistor varies its switching time steered by the control signal in relation to the analog input value a so called aperture uncertainty is applied.

Quantization

While sampling is working in the time domain, quantization affects the amplitude domain. During the hold time of the S/H the analog signal is converted into a digital representation. The ADC quantizes the analog signal by using one value out of a finite list of integer values to represent a given analog voltage level.

Figure **5.8** depicts the transfer function of an ideal 3-bit quantizer that maps the input signal into eight (2^3) digital output values. The transfer function corresponds to a staircase with equidistant steps. The width of the steps is referred to as code width. The code width signifies the smallest possible difference between two digitized values: the so called least significant bit (LSB). In this example 1 LSB equals 225 mV, as the full analog input range of 1.8 V is divided into eight LSB intervals each represented by a unique digital code.

The bottom graph in Figure **5.8** shows the quantization error. It is obtained by subtracting staircase from the input graph. The average error is 1/2 LSB. The error maximal values are found at integer multiples of the code width. The standard deviation error is $1/\sqrt{12}$ LSB.

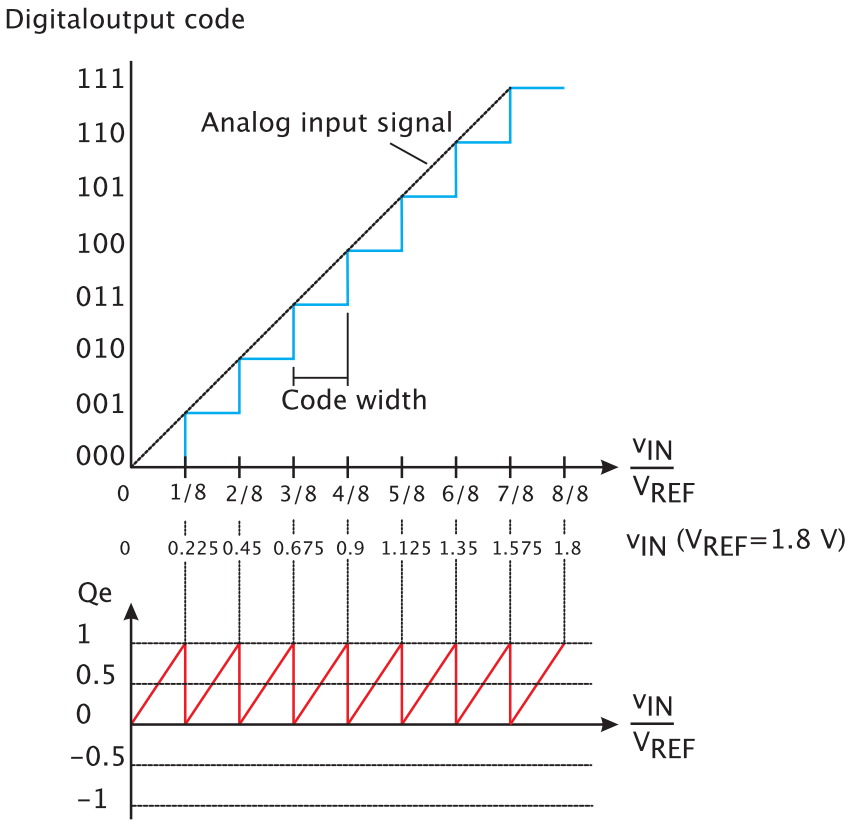


Fig. 5.8 *Transfer curve of an ideal ADC with its quantization error*

Range and Resolution

Naturally, only input values within certain voltage range, i.e. the input range, can be represented by a given ADC in a meaningful way. The maximum value that can still be converted is referred to as +full-scale, the respective minimum value as -full-scale. ADCs with a -full-scale of 0V are called unipolar. Conversely, bipolar ADCs have a symmetrical input meaning that the absolute values of +full-scale and -full-scale are identical. Nevertheless most ADCs offer reference inputs (V_{REF}) which allow the full-scale range to be adjusted (within the range of the power supply voltages).

Adjusting the input range of an ADC entails that its resolution, i.e. the voltage represented by the LSB, is subject to change, as well. Consequently, the resolution of an ADC is typically provided in terms of the number of bits offered by the ADC. A 16-bit ADC, for example, can resolve one part in 2^{16}=65536 of the set input range. However, other measures are frequently used to represent the ADC resolution (see Table **5.1**). E.g., the resolution of digital voltmeters is often given in terms of a number of digits: a 5-1/2 -digit

ISBN 978-3-527-33606-7

voltmeter on a 2V range resolves a range of +1.99999 V to -1.99999 V into a resolution of 0.00001 V.

Bits	Digits at a "Voltmeter"	Steps in Full scale range	Stepsize [ppm]	Dynamic Range [dB]
24		16 777 216	0.060	146
21.9	6 1/2	4 000 000	0.25	133
20.9	6	2 000 000	0.5	127
20		1 048 576	0.9	122
18.6	5 1/2	400 000	2.5	113
18		262 144	3.8	110
17.6	5	200 000	5	107
16		65 536	15	98
15.2	4 1/2	40 000	25	93
14.2	4	20 000	50	87
14		16 384	61	86
12		4 096	244	74
11.9	3 1/2	4 000	250	73
10.9	3	2 000	500	67
10		1 024	976	61
8.6	2 1/2	400	2500	53
8		256	3906	49
7.6	2	200	5000	47

Tab. 5.1 *Overview of different resolution and their corresponding values like "digits" and dynamic range*

Linear Errors

A systematic deviation of the ADC-transfer-function from the behavior of the ideal ADC that can be described by a linear function across the entire input range is referred to as linear error. Linear errors are a common occurrence and can be of substantial magnitude.

Generally two types of linear errors are distinguished: offset errors and gain errors. An offset error occurs as a result of a mismatch between the LSB and the first code transition, as shown in Figure **5.9**, left. Because offset errors are constant across the input range they can easily be corrected for by adequate calibration procedures. A gain error denotes the difference between the slope of the ADC-transfer-function form the behavior of the ideal

ADC, which has an average slope of 1 (see Figure **5.9**, right). Since this error is a systematic by definition, a correction could be implemented by a following microcontroller.

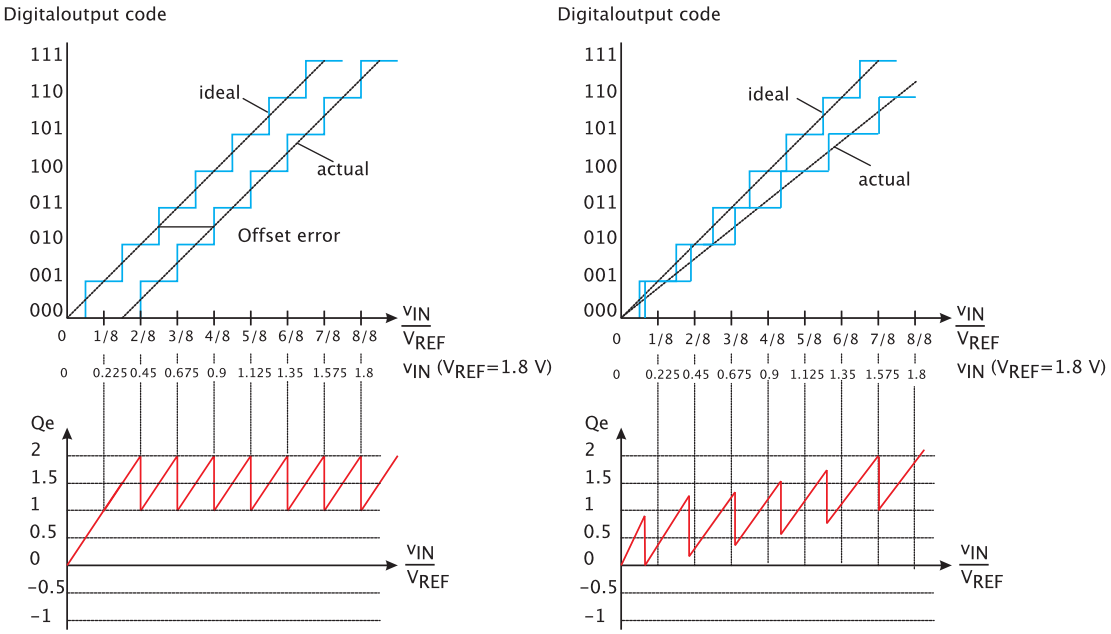


Fig. 5.9 *Left: Transfer curve of an ADC showing the offset error and the corresponding quantization error, Right: Transfer curve of an ADC showing the gain error and the corresponding quantization error*

Nonlinear Errors

The term differential nonlinearity (DNL) of an ADC denotes the difference between the actual code width of each step of the transfer function compared to the LSB of the ideal ADC. This is illustrated in Figure **5.10**. If the DNL exceeds 1 LSB is causes a non-monotonic transfer function which is known as a missing code.

The integral non-linearity is defined as the difference between the date converter code transition points and the straight line with all other errors set to zero. Therefor a straight line is drawn through the end points of the first and the last code transition. Another possibility to determine the INL is to take a look at the quantization error. There the INL will have the value of the highest peak minus the 1/2 LSB of the quantization error of an ideal ADC.

ISBN 978-3-527-33606-7

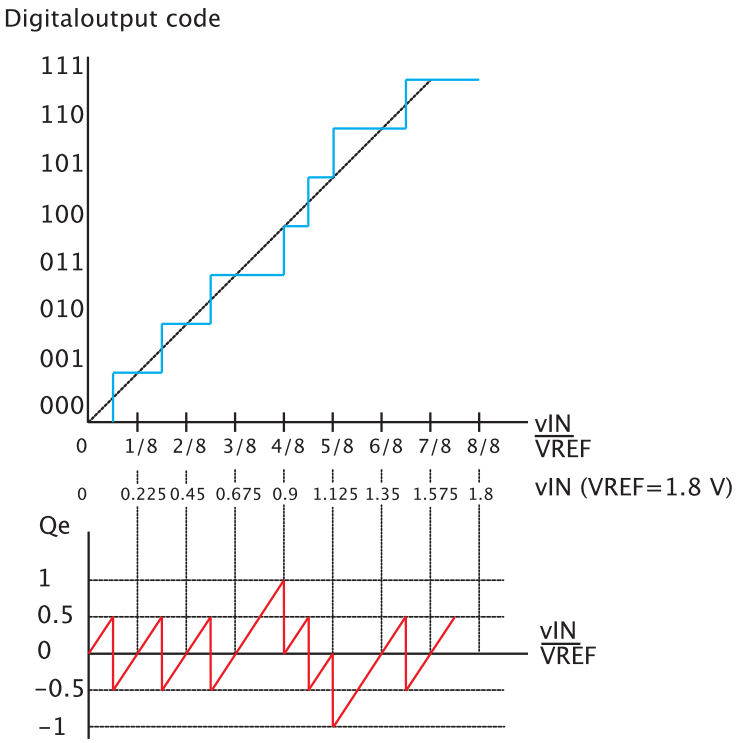


Fig. 5.10 *Transfer curve of an ADC indicating a nonlinear error and the corresponding quantization error*

Dynamic Range

The signal-to-noise ratio (SNR) of an ADC represents the ratio between the largest RMS input signal, v_{in}, and the RMS value of the noise added by the ADC, v_{noise}:

$$SNR = 20\log\left(\frac{v_{in}}{v_{noise}}\right) \tag{5.3}$$

If the input signal is a sine wave with a peak-to-peak value equal to +full-scale and –full-scale the v_{in} given by

$$v_{in} = \frac{v_{fullscale}}{2\sqrt{2}} = \frac{V_{ref}}{2\sqrt{2}} = 2^N \frac{V_{LSB}}{2\sqrt{2}} \tag{5.4}$$

where V_{LSB} indicates the voltage value of the least significant bit. The RMS of the noise voltage of an ideal ADC is given by its quantization error. In respect to Figure **5.8** the value can be calculated as

$$v_{noise} = \sqrt{\frac{1}{V_{LSB}} \int_{-0.5V_{LSB}}^{0.5V_{LSB}} V_{LSB}^2 dV_{LSB}} = \frac{V_{LSB}}{\sqrt{12}} \tag{5.5}$$

The SNR of the ideal ADC is thus given by:

$$SNR = 20\log\left(\frac{2^N V_{LSB}}{2\sqrt{2}}\right)\frac{\sqrt{12}}{V_{LSB}} = 6.02N + 1.76 \tag{5.6}$$

This equation denotes an important relation between the resolution and the SNR of an ideal ADC. An ideal 16-bit ADC thus has a signal-to-noise ratio of 98.08 dB. For non-ideal ADCs this concept is often reversed to obtain a meaning full measure for the resolution of the device. In this approach the SNR of the ADC is measured and the so called effective number of bits (ENOB) is calculated as

$$ENOB = \frac{SNR - 1.76}{6.02} \tag{5.7}$$

Power Dissipation

The power dissipation of ADC depends on multiple parameters. Commonly, the mean of the power dissipation has a proportional dependency of the sampling rate, mainly basing on transistor switching effects (considering CMOS circuit designs), but not limited to. The power dissipation depends significantly on the data converter architecture and the par-

ISBN 978-3-527-33606-7

ticular circuit implementation, discussed in the next sub-section. Apart the usable resolution and the source-to-noise ratio the power consumption of ADC components is a central design criteria in the deployment in low-power and very-low-power sensing systems.

5.1.3 Data Converter Architectures

Considering that currently thousands of converters are available on the market, the selection of the architecture most suited for the given task is an essential part of circuit design. In this selection process several aspects have to be considered. Sigma-delta ADCs, e.g., achieve a high resolution while the corresponding sample-rate is comparatively low. Therefore sigma-delta ADCs are ideal for application in sensor signal processing, where conversion speed is not essential. Conversely, flash converters and pipeline architectures are the fastest ADC available, yet due to aspects such as chip area consumption they are limited to lower resolution. In Figure **5.11** the most common ADC architectures are compared in terms of their resolution and sample-rate. Some more details can be found in **[KES05]**, and design aspects are discussed in **[PLA05]**. As shown inthe next sections, ADC designs consist of analog and digital parts. A deeper discussion on mixed-signal circuit design can be found in **[BAK10]**.

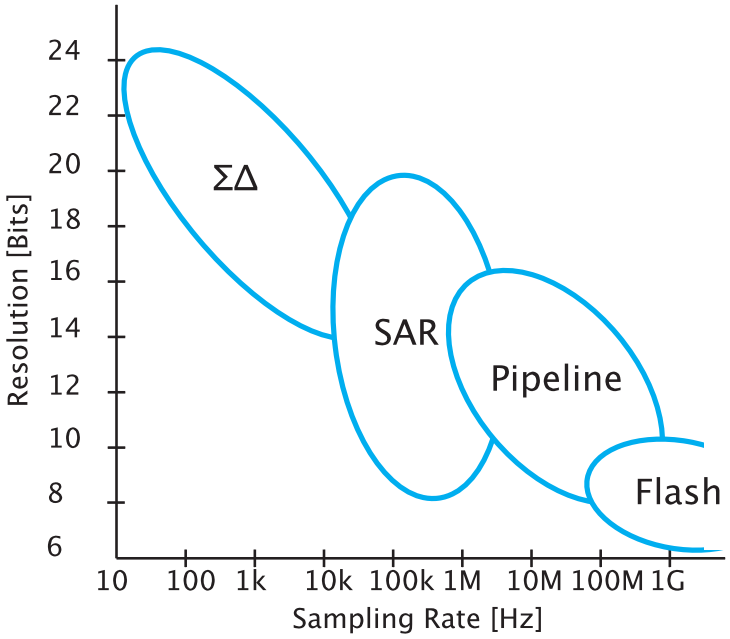


Fig. 5.11 *Different ADC architectures and their common usage in terms of resolution and sample rate*

The basic component of every ADC is the comparator. It can be seen as a one-bit ADC as it has two analog inputs and one digital output. The output is a digital 1, if the voltage at the non-inverting input of the comparator exceeds the voltage level at the inverting input. Otherwise, the output is a digital 0. Another functional part is the reference circuit that determines the relationship between an analog input-value and the corresponding digital output. The performance of the reference circuit directly impacts on the magnitude of non-linear errors, such as DNL and INL. Furthermore, every ADC offers the possibility to set the input range by means of a reference Voltage.

Integrating ADC

Integrating converters are used for high-resolution and low-speed applications. A schematic overview of an integrating ADC implemented in the so-called dual-slope architecture is illustrated in Figure **5.12**.

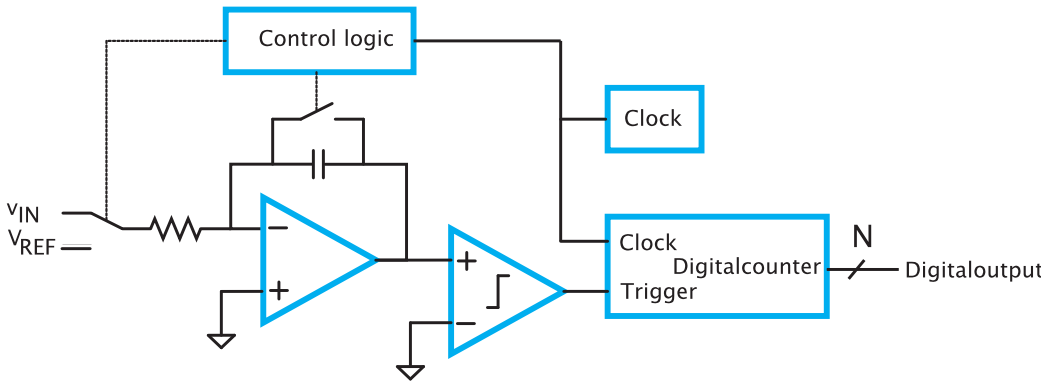


Fig. 5.12 *A dual-slope integrating architecture*

The basic components of this architecture are an integrating amplifier followed by a comparator and a digital counter. At the beginning of the conversion process, the capacitor of the integrator is discharged by the control logic. At t=0 the analog input signal V_{in} is connected to the input of the integrator and the capacitor is charged. After a constant time the input is switched to the reference voltage, V_{REF}. The voltage at t_1 is directly proportional to the analog input voltage. Because the reference voltage is negative, the capacitor will discharge at a rate proportional to the reference voltage. The digital counter measures the time from t_1, until the capacitor is fully discharged and the comparator will switch its output at t_2. An overview of the charging and discharging process is given by Figure **5.13**.

ISBN 978-3-527-33606-7

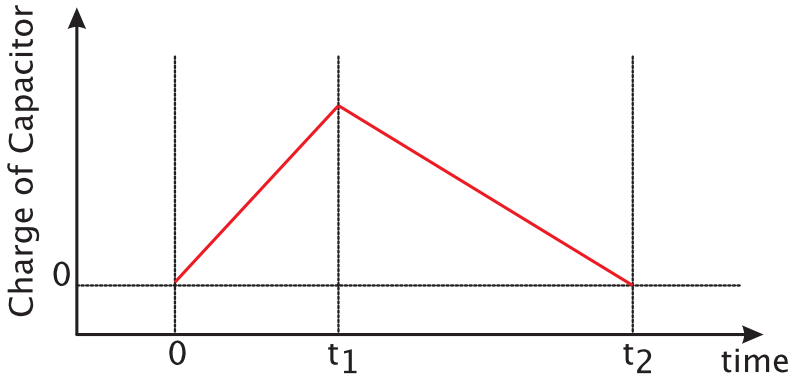


Fig. 5.13 *Charge of the integrating capacitor vs. time*

Assuming an ideal capacitor, the ratio between the discharge time and the charge time is proportional to the ratio of the input voltage to the reference voltage. Consequently, the digital counter value is proportional to the analog input voltage, as well.

It is important to note, that the input voltage is averaged during the charging process of integrating ADCs. Therein lays an important difference with respect to ADCs that employ sample-and-hold techniques. The averaging can be advantageous as it tends to reject periodic noise. This is best utilized by using an integration time that is an integer multiples of the AC line period of 50 Hz or 60 Hz, so that interferences from the power line are canceled.

Sigma-delta Converters

The sigma-delta (ΣΔ) ADCs have become a popular architecture choice. ΣΔ-ADCs often have resolutions above 16-bit at sample-rates below 100 kS/s. The high resolution is achieved by oversampling, noise shaping and decimation. These the three working principles are illustrated in Figure **5.14**.

ISBN 978-3-527-33606-7

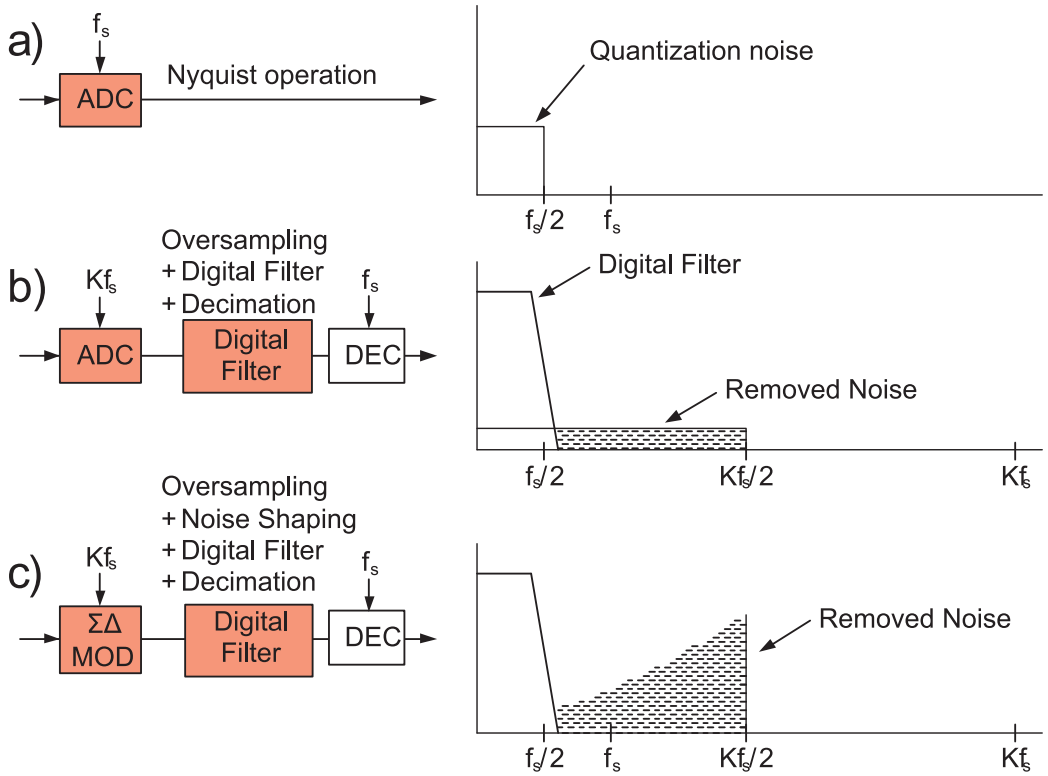


Fig. 5.14 *Overview of the principles of oversampling, digital filtering, noise shaping and decimation*

Figure **5.14**a) shows the distribution of quantization noise in the frequency domain for a sampling ADC operating at sampling rate of f_s to fulfill the Nyquist-criteria for signals with frequencies up to $f_s/2$. If a (much) larger sampling rate is used than strictly required to fulfill the Nyquist-criteria the rms quantization noise is spread over a wider bandwidth; i.e. from DC to $Kf_s/2$, where K is the oversampling factor and f_s the required sampling rate. This entails, that the application of a digital low-pass filter removes a fraction of the quantization noise thus increasing the SNR was well as the ENOB (see Figure **5.14**b).

However, if one were to only use oversampling to increase the resolution, the oversampling factor K would need to be 2^{2N} for a gain of N-bits of resolution. ΣΔ-ADCs mitigate the requirement for unreasonably large oversampling factors by shaping the quantization noise to higher frequencies (see as Figure 14c). This is achieved by a so-called ΣΔ modulator. Based on the state of a 1-bit ADC, i.e. a comparator, a 1-bit DAC signal is generated. The DAC signal is summed with the input and fed to an integrator, which in turn is directly connected to the comparator thus creating a negative feedback loop (see Figure **5.15**.).

ISBN 978-3-527-33606-7

The input voltage is represented by the serial output stream of the comparator as the feedback loop forces the average DC voltage on the output of the 1-bit DAC to be equal to V_{IN}. If one, e.g., assumes a DC of V_{IN}=0 at the input of a bipolar ΣΔ-ADC, the integrator is constantly ramping up and down and the ratio between high and low states at the output of the comparator will be 1. If the input signal increases towards $+V_{REF}$, the number of high states in the serial output stream of the comparator will increase, as illustrated in Figure 16. Conversely, if the input signal shifts towards $-V_{REF}$, the number of low states will increase, thus reducing the average voltage.

The subsequent digital filter and the decimator process the bit stream to the final output at the sample frequency of f_s. The fact, that the digital output filter reduces the bandwidth entails that the output data rate can be reduced by the decimator, without violating the Nyquist-criteria. The decimator only passes every Xth result to the output while the rest is discarded. X can be any integer value as long as the output data rate is still larger than twice the input signal bandwidth.

To obtain a meaningful result at the output of the ΣΔ ADC a large number of samples should be averaged as the dynamic range increases as more samples are averaged. E.g. a digital low-pass filter that averages four samples can only assume four states indicating a resolution of 2-bit. If the filter is averaging every 16 samples the resolution yields 4-bit and so forth. Figure **5.16** shows the output of the integrator and comparator for an input DC voltage of $V_{IN}=V_{REF}$ / 2. The respective output values for the 2-bit and 4-bit (not shown) ΣΔ ADC are 3/4 and 12/16, respectively.

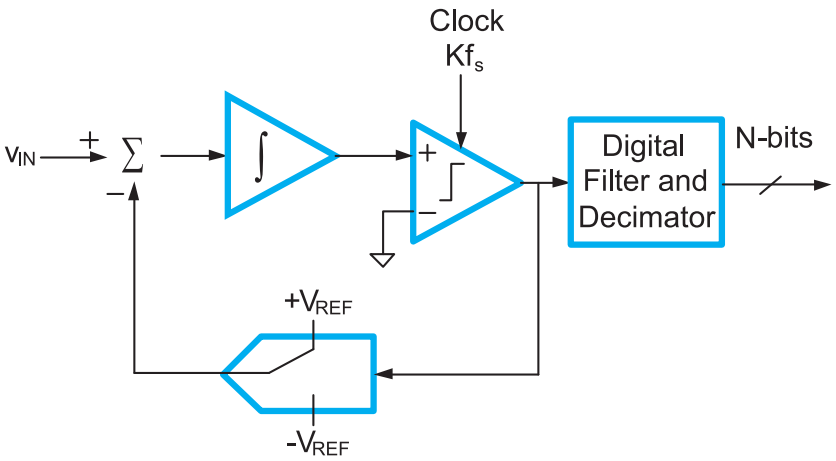


Fig. 5.15 *First-order sigma-delta ADC*

ISBN 978-3-527-33606-7

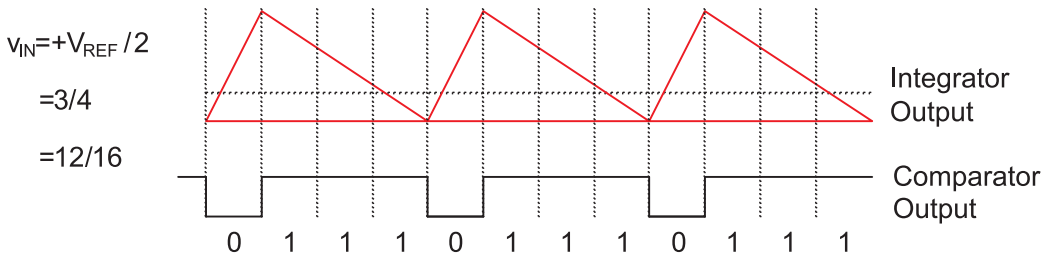


Fig. 5.16 *Sigma-delta waveforms at a certain analog input voltage*

The concept of the ΣΔ ADCs noise shaping is best explained by considering the frequency-domain. The integrator of the ΣΔ modulator is working as an analog low-pass filter with the transfer function $H(f)=1/f$. The 1-bit DAC generates the quantization noise Q which is fed back to the input of the integrator as indicated in Figure **5.17**.

For a given input signal *IN* the signal after the summing block (i.e at the input of the integrator) is be *IN* - *OUT*. This signal is multiplied by the transfer function of the integrator. Finally the quantization noise is added resulting in the following equation:

$$OUT = \frac{1}{f}(IN - OUT) + Q \tag{5.8}$$

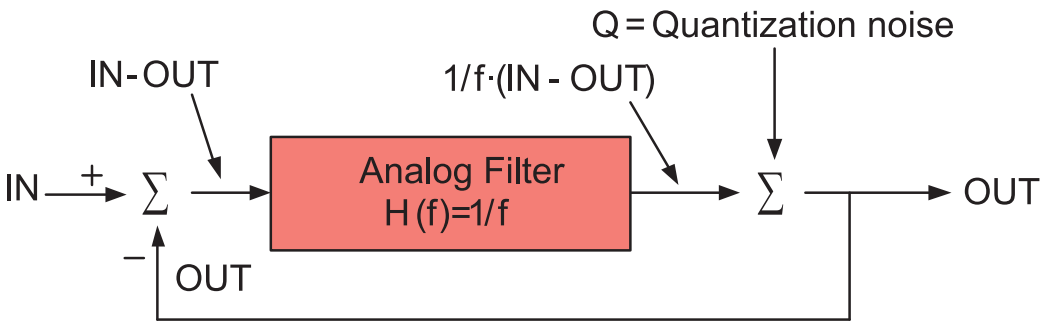


Fig. 5.17 *Simplified frequency domain model of a sigma-delta modulator*

Solving this equation for *OUT* results in

ISBN 978-3-527-33606-7

$$OUT = \frac{IN + Qf}{1 + f} \tag{5.9}$$

If the frequency of the input signal approaches zero, the output voltage consists entirely of the input voltage without any quantization noise. At higher frequencies the amplitude of the input signal will decrease while the quantization noise approaches the value of Q. In essence, the integrator has a low-pass effect on the input signal while having a high-pass effect to the quantization noise. Therefore the integrator working as an analog filter is performing the noise shaping function of a ΣΔ ADC. To achieve even higher resolutions at the same oversampling frequency higher-order ΣΔ modulators can be implemented. By using higher order analog filters the effect of the noise shaping is increased due to the higher attenuation of the filter. An example for a second-order ΣΔ ADC is given in Figure **5.18**.

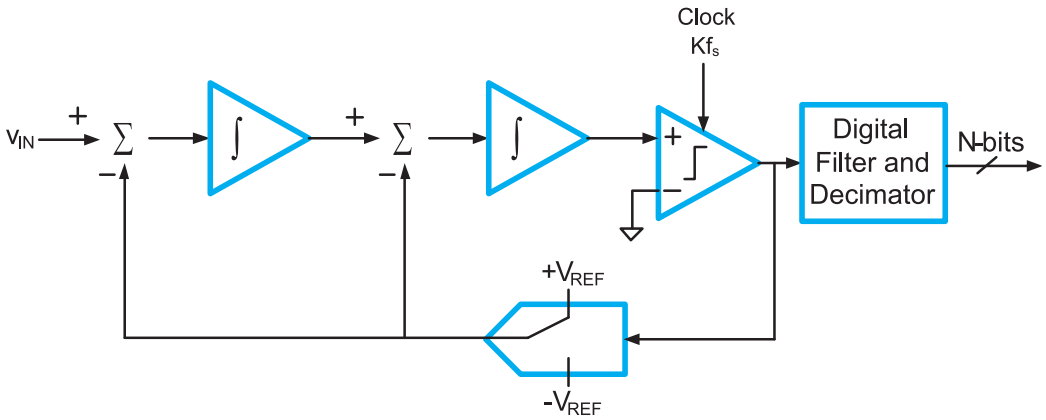


Fig. 5.18 *Second-order sigma-delta ADC*

 ISBN 978-3-527-33606-7

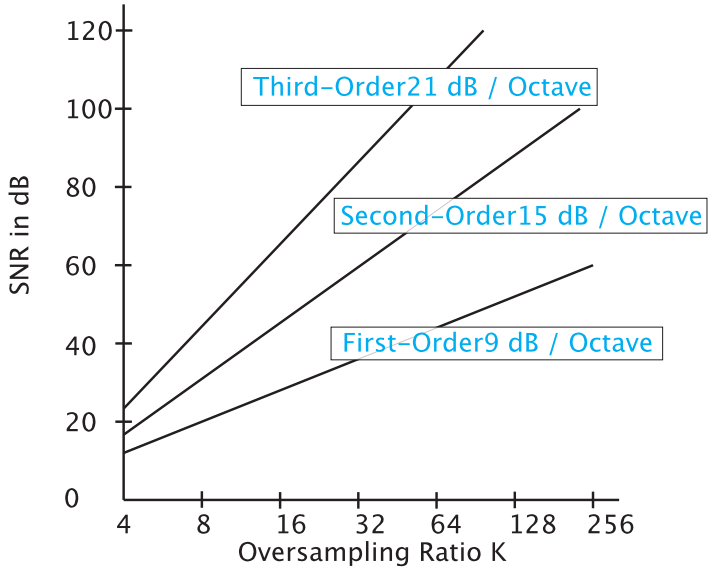


Fig. 5.19 *SNR vs. Oversampling ratio for different sigma-delta modulator orders*

In Figure **5.19** the relationship of a ΣΔ ADC to the achievable SNR is shown. If a resolution of 12-bit is required the necessary signal to noise ratio (SNR) of the ADC has to be 74 dB. This is, for example, achieved, using third-order ΣΔ modulator with an oversampling factor of at least 16. In order to achieve the same resolution with a first-order ΣΔ modulator, the factor would need to be approximately 512. Since high clock frequencies in a mixed-signal integrated circuit can cause severe difficulties regarding the performance of the analog circuit blocks, a trade-off between more complicated higher-order ΣΔ architectures and the oversampling ratio has to be found.

Successive approximation ADC

The successive-approximation ADC is by far the most popular architecture concerning data-acquisition systems. Having resolutions between 8 and 18 bit while the sample-rate can be greater than 1 mega sample per second (MSPS). In addition, they are low in cost due to the low chip-area consumption.

ISBN 978-3-527-33606-7

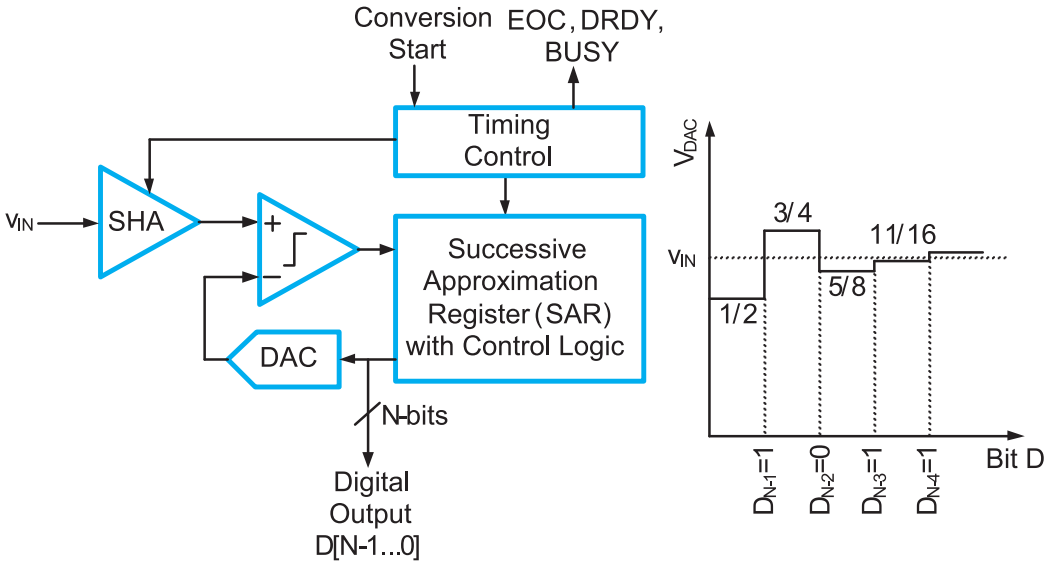


Fig. 5.20 *(Left) Basis Architecture of a successive approximation ADC; (Right) weighting process for a given analog input signal*

At the left side of Figure **5.20** the basic components of a SAR-ADC are shown. At the start of the conversion process, the S/H amplifier (SHA) is in hold mode. All bits in the successive approximation register (SAR) are set to 0 except the MSB, which is set to 1. The SAR output drives the digital analog converter (DAC) to apply the first reference voltage to the comparator. Therefore, analog voltage at the output of the DAC is always $V_{DA} = V_{REF} / 2$ at the beginning of a conversion cycle. If the output of the DAC is greater than the analog input, the MSB in the register will be cleared, otherwise it is left set. The next most significant bit is set and the process continuous until the LSB is reached. Consequently, N comparison operations are required, in order to convert the analog input signal into an N-bit digital word. An example for this conversion process is shown on the right side of Figure **5.20**.

The conversion time and the sample rate of a SAR-ADC, are given by

$$T_{conversion} = NT_{setting} = \frac{1}{f_s} \tag{5.10}$$

It is evident, that the speed of the ADC is limited by the settling time, $T_{settling}$, of the DAC. It is important to note, that the DAC may need a longer time to settle its output voltage within the tolerances of 1/2 LSB, if the resolution is high. This entails, that the conversion time does not necessarily scale linearly with N.

One of the most popular successive approximation architectures uses the binary-weighted capacitor array as its DAC. The reference voltage of the DAC is based on the amount of charge on each of the DAC capacitors. Figure **5.21** shows an N-bit architecture.

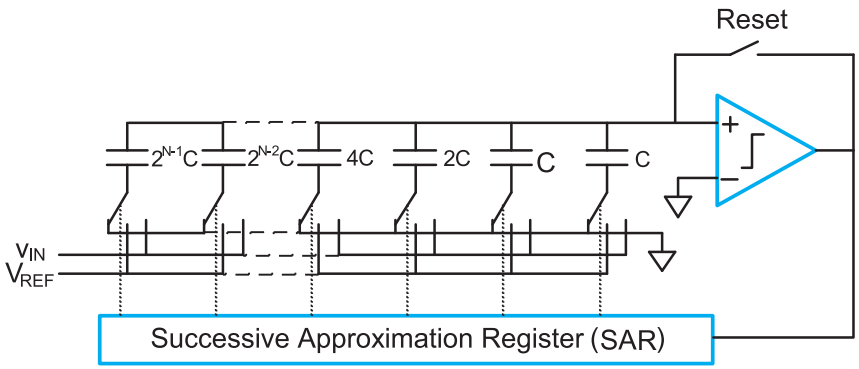


Fig. 5.21 *A charge redistribution architecture using a binary-weighted capacitor array*

The binary-weighted capacitor array is sampling the analog input voltage such, that no sample-and-hold circuit is required. Before the conversion starts the reset switch is closed and the capacitor array is discharged. As long as the reset switch is closed the comparator acts as a unity gain buffer. In this state the capacitor array charges to the offset voltage of the comparator performing an offset cancellation. In order to start the conversion process, V_{in} is connected to the input nodes of the entire array. The array is charged while the reset switch is still closed, holding its side of the capacitor array at virtual ground. The reset switch is opened while the input of the capacitors is switched from back to ground, simultaneously. As a result, the voltage at the comparator input is now V_{offset} - V_{in}.

To initiate the first conversion step the input of the MSB capacitor (i.e. the capacitor with the largest capacitance) is switched to V_{REF}. The current voltage at the comparator input is now

$$V_{incomparator} = V_{offset} - V_{in} + D_{N-1}\frac{V_{REF}}{2} \tag{5.11}$$

where D_{N-1} is the output of the comparator, i.e. the MSB. If the comparator output is high the input of the MSB capacitor remains connected to V_{REF}, otherwise it is switched back to ground. Subsequently, the second largest capacitor is switched to V_{REF} and the voltage at the comparator input array becomes

ISBN 978-3-527-33606-7

$$V_{incomparator} = V_{offset} - V_{in} + D_{N-1}\frac{V_{REF}}{2} + D_{N-2}\frac{V_{REF}}{4} \tag{5.12}$$

The conversion process continuous until every bit in the successive approximation register is tested. The input voltage of the comparator at the end of the conversion process can be described as

$$V_{incomparator} = V_{offset} - V_{in} + D_{N-1}\frac{V_{REF}}{2} + D_{N-2}\frac{V_{REF}}{4} + \cdots + D_1\frac{V_{REF}}{2^{N-1}} + D_0\frac{V_{REF}}{2^N} \tag{5.13}$$

where N represents the resolution of the ADC.

Flash ADC

Flash ADCs, or parallel ADCs, use a large number of comparators simultaneously for the signal conversion, which enables extremely fast conversion times. The schematic of a 3-bit Flash ADCs is shown in Figure **5.22**. It can be deduced from this figure, that an N-Bit Flash ADC requires 2^N-1 comparators. Furthermore, 2^N resistors are required, as each comparator needs its own reference voltage, which is supplied be the string of resistors. The resistors are equal, with the exceptions of the first and the last resistor in the string. Thus all neighboring nodes differ by 1 LSB.

The output of all comparators is often referred to as a thermometer code, because a certain input voltage causes the output of all comparators with a lower reference voltage to transition to a high state. A digital encoder converts this data into the corresponding binary code. Because only the comparators and the digital encoder are limiting the sample-time fast converters can be realized. This advantage is counterbalanced by the need to double the chip-area to gain one bit of resolution. Therefore, flash converters rarely exceed 8-bit resolution.

One approach to limit the number of necessary comparators is the implementation of a two-step flash ADC. The working principle of a two-step flash ADC is illustrated in Figure **5.23**. The converter consists of two flash ADCs where to one converter estimates the most significant bits (MSB). The result of this *coarse conversion* is converted back into an analog signal and subtracted from the input signal. The remainder is converted by the second flash ADC. This ADC delivers the least significant bits, as its full range corresponds to the LSB of the coarse converter. The amount of comparators used by a two-step flash converter is given by $2(2^{N/2}-1)$. For example a 10-bit Flash converter requires 1023 comparators while a two-step flash ADC requires only 62.

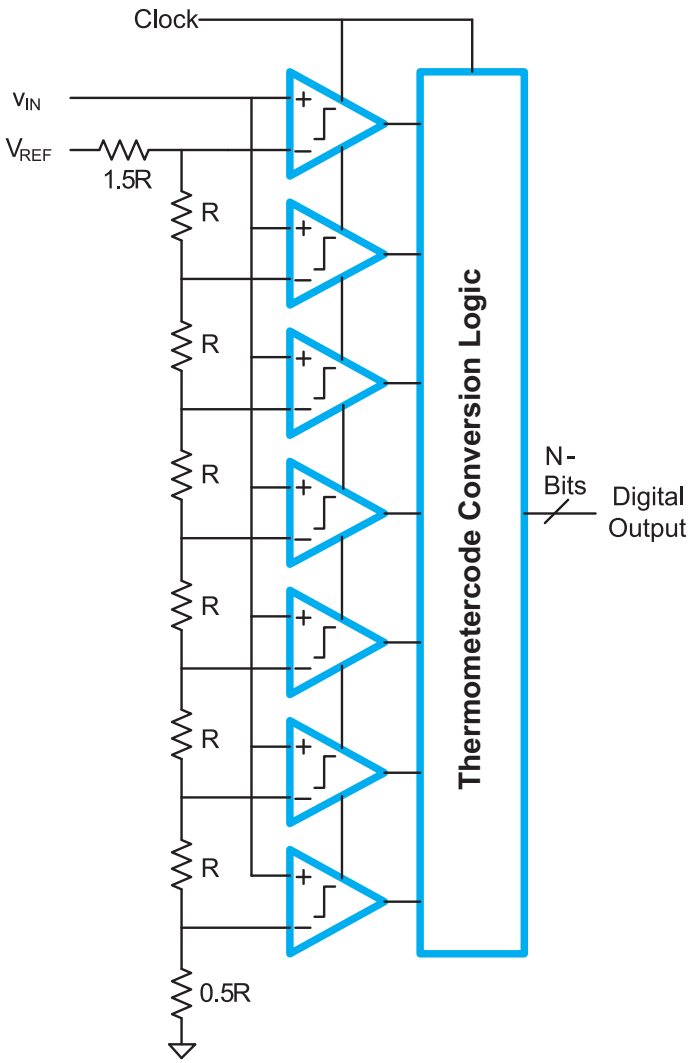


Fig. 5.22 *A 3-bit parallel Flash Converter*

ISBN 978-3-527-33606-7

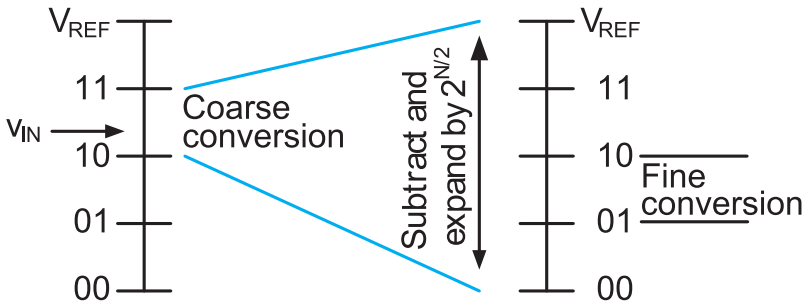


Fig. 5.23 *Principle of coarse and fine conversion using a two-step Flash ADC*

5.1.4 Low-power ADC Designs and Power Classification

In Fig. **5.11** different converter architectures were classified by resolution and sampling rate domains. But power dissipation is the third central parameter in the design of material-integrated sensing systems demanding for low-power devices. Though there is a relation between the resolution, sampling rate, converter architecture, circuit technology, and the power dissipation of an ADC, it is difficult to give estimations of expected power dissipation by these parameters. There are some rough estimations to classify and quantify the power dissipation of different ADC architectures and in relation to the bit-resolution and sample rate. In **[SUN09]** it was stated that the minimum power dissipation (lower bound) for data sampling of nyquist ADCs (no oversampling) that can be achieved is proportional to the sampling frequency f_S and the power of the bit-resolution N:

$$P_S \sim f_S 2^{2N} \tag{5.14}$$

Similar discussion about the lower limits of ADC are discussed in **[MUM06]**, analyzing the data sampling circuit of ADCs.

In Fig. **5.24** the normalized power dissipation of different ADC architectures is shown, depending on the effective resolution (SNR bits) and giving a comparison with the estimated minimal data sampling power dissipation bound from Eq. **5.14** (based on data from **[MUR07]**). As expected, there is a monotonic increase of the power dissipation in relation to the bit resolution. It is interesting that the current ADC designs have much higher power requirements than the predicted minimum power bounds. Surprisingly, the normalized power dissipation of flash ADC architecture scales better than for pipelined or SAR (other) architectures. Due to the power dependency from bit-resolution attempts were made to reconfigure the bit-resolution at run-time, e.g., **[YIP11]**, proposing a 5-10 bit SAR ADC. Furthermore, the supply voltage can be varied between 0.4 and 1V, further lowering the power dissipation with low sample rates. Leakage effects of the electronic circuit become dominant at sample rates below 1-2 kS/s.

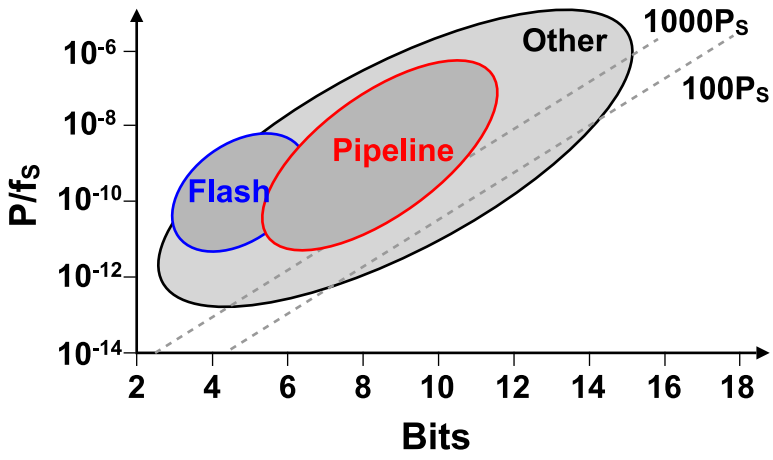


Fig. 5.24 *Normalized power dissipation of different ADC architectures depending on the effective resolution (SNR bits) and comparison with the estimated minimal power dissipation bound from Eq.* ***5.14****.(based on* **[SUN09]** and *data from* ***[MUR07]****)*

In **[SCO03]**, an ultra-low power ADC architecture and implementation was proposed using the SAR conversion method. They achieve a power consumption in the µW range by providing 8bit resolution and sample rates up to 100kS/s (1V supply voltage). The standby power consumption could be achieved in the pW range. A conversion requires an energy about 30pJ. High-speed ADC can achieve power consumptions down to the mW range, based on SAR and pipelined SAR principles (e.g., a 9-bit 150MS/s SAR ADC **[YIN13]**).

5.1.5 Moving-Window ADC Approach

Resistive sensors, like strain-gauge sensors, provide only a small relative change in resistance in the order of 1% resulting from a change of applied load in the considered operating range of the sensor. Using bridge configuration, providing a differential signal, require compensated sensors with small tolerances in strain and zero-load resistance parameters, actually not applicable to sensorial materials using, for example, printed sensors.

Assuming only one non calibrated and uncompensated resistive sensor, a zooming window approach (see Equation **5.15**) can be used to match an initially unknown sensor to the measurement system preserving a high and full-range resolution, shown in Figure **5.25**.

$$W(s) = k(s - off) \tag{5.15}$$

ISBN 978-3-527-33606-7

with *W* is the window, *k* the zoom factor, and *off* the shift of the window.

The data processing performs an initial (or periodically repeating) auto-calibration finding the centre of the operational window by using fast settling successive approximation, shown in Algorithm **5.1**. The zoom factor *k* is kept fixed (determined by the amplifier gain settings of the ADC/DAC sections), and only the offset of the window is calibrated.

Alg. 5.1 *Auto calibration using successive approximation*

```
1  sar ← DIGITALRANGE/2;
2  DAC1 ← GAIN0, DAC2 <- 0;
3  WHILE sar <> 0 do begin
4    IF ADC > DIGITALRANGE/2
5      THEN DAC2 ← DAC2 + sar ELSE DAC2 ← DAC2 - sar;
6    sar ← shift_right(sar,1); end;
7  off ← DAC2-DAC1;
```

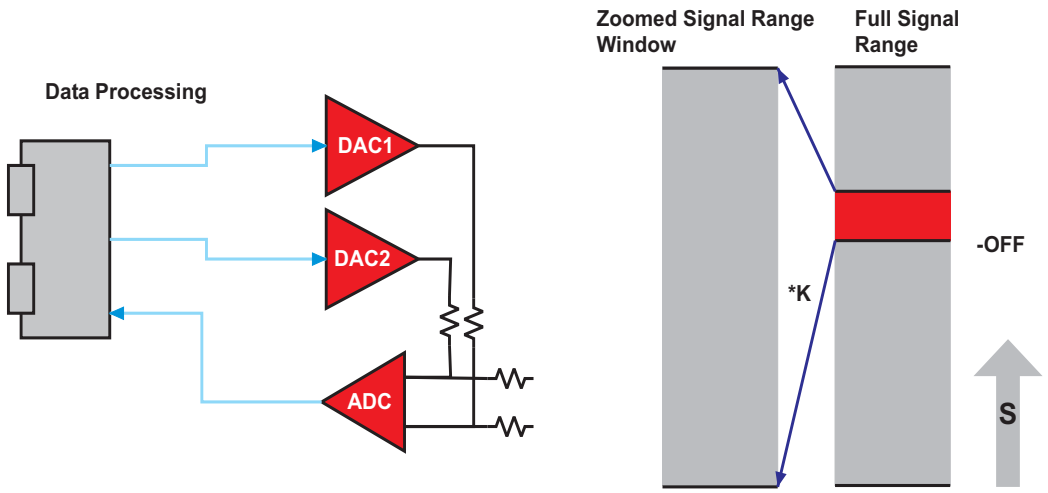


Fig. 5.25 *Zooming and Moving Window ADC Approach, used, for example, with resistive Strain-gauge sensors (ADC: Analogue-to-Digital Converter with differential input, DAC: Digital-to-Analogue Converter).*

Reconfigurable ADC supporting different resolutions can support the window approach significantly in presence of varying bandwidth and dynamic ranges. For example, in **[YIP11]** a reconfigurable SAR ADC architecture was proposed that provides a bit-resolution range from 5 to 10 bits, supporting an optimal dynamic adaption if used in conjunction with the window approach. Hence, the extended optimized dynamic range adaption using resolution-configurable ADC with moving and zooming windows enables

the design of very-low-power sensor networks, a prerequisite for, e.g., material-integrated sensing systems and self-supplied sensing systems with very limited available energy.

ISBN 978-3-527-33606-7

5.2 Digital Real-Time Data Processing with Embedded Systems [Bosse, Lehmhus]

5.2.1 Levels of Information

On a long term scale, material-integrated sensor systems will become common. Structural materials thus equipped will be able to judge their own state irrespective of the loads they experienced, and will not lose this ability even if damage has been incurred. Structures of this kind will show autonomy at least in their evaluation of their structural state, and the adaptation of their self-perception they will base on this evaluation

The field of structural monitoring and control has been evolvled in than 20 years significantly, leading to a classification of systems originally proposed by Rytter **[RYT93]** and recently summarized by Lehmhus et al. (**[LEH13B]**, see Fig. **5.26**)

The deployment of embedded systems for sensor data processing can be found in the following areas:

Structural Monitoring and Control

Automated gathering of information on mechanically loaded structures has originated in civil engineering, with bridges or skyscrapers as the primary objects of supervision **[CHO99][PEE09][SUN10]**. Sensor node size is typically macroscopic here, and outward attachment to structural members sufficient. However, the “need for speed” is clearly there. Miniaturization in combination with new developments in aerospace engineering, not the least the broader introduction of composite materials in replacement of metals, has fueled interest in the technology in this area for more than a decade **[REN01]**. At the very end of these trends are morphing structures that facilitate fly-by-feel concepts in unmanned aerial vehicles (UAV). These again strongly stress the interest in immediate availability of information. Recently, maintenance issues associated with offshore wind energy plants, which also rely heavily on composite materials e.g. for rotor blades, have created similar interest here.

Structural Health Monitoring

Structural Health Monitoring (SHM) adds the ability to derive not just loads, but also their effects from sensor data. Boller **[BOL09]** gave a definition of a SHM system: *A SHM is the integration of sensing and possibly also actuation devices to allow the loading and damaging conditions of a structure to be recorded, analyzed, localized, and predicted in a way that nondestructive testing (NDT) becomes an integral part of the structure and a material*

Load Monitoring

A load monitoring system (LM) can be considered as an incomplete subclass of a full SHM system, which provides spatial resolved information about loads (forces, moments, etc.) applied to a technical structure. When implemented in robot grippers

[TRA12], such systems can improve grasping performance via feedback control. Another application can be found in long distance robot manipulator structures to improve position accuracy **[MAV97]**. In aerospace industry, load monitoring systems as first step towards structural health monitoring were initially implemented in military aircraft like the Eurofigther TYPHOON **[HUN01]**, but have meanwhile entered the civilian market in parallel to their ongoing maturing into true SHM systems **[RUL13][SAN13]**. Real-time capabilities of the load and structural monitoring system can be essential.

Structural control, adaptive and morphing structures

The term structural control implies a material-inherent capability of changing structural characteristics in response to, and countering the effects of external loads. It is thus beyond mere sensing and thus beyond the scope of systems addressed within this text. These however form the necessary basis of farther-reaching adaptive systems.

Besides alleviating structural loads, structural control-like systems are foreseen to facilitate a fly-by-feel approach for autonomous flight of unmanned aerial vehicles (UAVs) **[SAN13]**. At a second glance, this scenario can be seen as linked to robotic tactile sensing: In both cases, a major extension of perceptive capabilities at the interface between system and environment is foreseen to support interaction with the latter, and based on it, allow for increased autonomy.

Tactile Sensing

Tactile sensing systems provide extrinsic perception for robots and robotic applications **[CAN10]**, via systems commonly designated as smart or artificial skin. Basically they deliver spatial resolved information on forces applied to an extended but limited surface region, for example, of robot connection elements or finger tips of a robot hand **[DAH07] [VID11]**. The finger tip example is probably what comes to mind first when thinking of robotic tactile sensing. However, covering areas other than a fingertip with a smart skin capable of interpreting tactile sensor data bears additional advantages: Sensor networks of this kind can provide the robotic system with much finer information about – voluntary as well as involuntary – contacts with its surroundings than state-of-the-art joint-integrated load monitoring ever will. This can be exploited to enhance safety levels in robot-robot or human-robot cooperation, but just as well to endow the robot with the capability to monitor its own actions, profiting from their success or failure via reinforcement learning. The concept has been applied to humanoid robots in the form of tactile sensor arrays to help monitor, control and ultimately improve their strategy to dynamically stand up from lying on their back. Such examples are merely a glimpse at the full potential of material-integrated sensor systems. From the above, many development trends can easily be extrapolated, like the simple change of perspective that turns robotic smart skins into new kinds of tactile user interface for countless purposes. The real-time capability of the tactile sensing system is fundamental.

ISBN 978-3-527-33606-7

Fig. 5.26 *Algorithmic and information level hierarchy in SHM and LM systems*

5.2.2 Algorithms And Computational Models

The raw sensor output of a Structural Monitoring or Tactile Sensing system reflects the lowest level of information. Both use cases incorporating sensor networks with sensor nodes acquiring sensor data of different dimension, regarding the spatial and the temporal dimension, starting from a single strain-gauge sensor delivering a scalar sensor value up to temporal distributed wave receiver data. Beside technical aspects of sensor integration of the main issue in those applications is the derivation of a mapping function $F_m(S)$ which basically maps the raw sensor data input S, in general a n-dimensional vector consisting of n sensor values (usually sampled from electro-mechanical transducers like piezo-electric sensors, or from electro-optical transducers like fiber-optic sensors **[BAL06]**), to the desired information I, in general a m-dimensional result vector:

$$F_m(S):\vec{S}\rightarrow\vec{I}$$
$$\vec{S}=(s_1,s_2,...),\vec{I}=(i_1,i_2,i_3,...) \tag{5.16}$$

The function F_m can be considered as a feature extraction function, and the output vector I can be classified in

- A condensed senor data vector retrieved by sensor data fusion, that means $S=(s_1, s_2, .. , s_n) \rightarrow I=S'=(s'_1, s'_2, .. , s'_m)$ with $m < n$, to improve the confidence of the measured sensor data;
- A spatially resolved load vector with an estimation of the load (forces) applied to a structure under test, for example, used in artificial skin **[DAH10]** or robot manipu-

lator **[FRA08]** applications, providing the estimated center location (coordinates) of the region and the strength information of the load, e.g. $I=(x,y,z,v)$;

- Or at higher abstraction level a proposition vector of detected features and patterns f_i, for example, a damage-sensitive feature (structural defect) or a decision for maintenance, e.g. $I=(f_1, f_2, f_3,..)$ with $f_i=[0,1]$.

This mapping function is usually built up by functional composition, with different functions ranging from low-level to complex high-level signal processing algorithms:

$$\begin{aligned} &F_m(S) = f(g(h(...(m(S))))) : S \to I \rightleftarrows \\ &S \mapsto m \mapsto ... \mapsto h \mapsto g \mapsto f : \\ &(S \to \alpha) \to (\alpha \to \beta) \to ... \to S \to I \end{aligned} \tag{5.17}$$

That means the signal processing system is composed of initially independent functions *f, g, h,...* (which can be considered as higher order functions treating arguments as functions and allowing partial evaluation) arranged in a pipeline chain, each performing the calculation of an intermediate result based on either the original sensor data ***S*** or already computed intermediate data ($\alpha \to \beta$). From a data processing point of view these functions are operational units which can be modeled with processes with data dependencies, requiring synchronized inter-process communication, discussed in Chapter **6**.

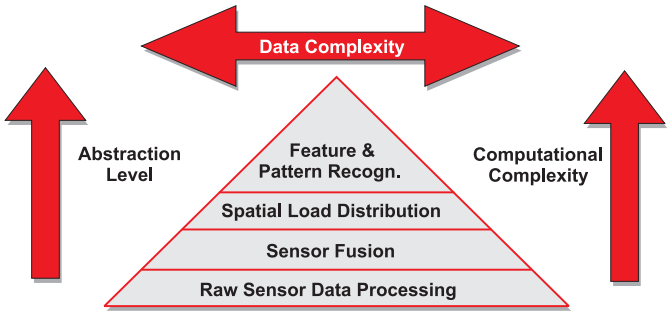


Fig. 5.27 *Data processing classes common to SHM, LM, and TS systems*

Basically there are two different information extraction approaches: I. those based on an accurate mechanical and numerical model of the technical structure and the sensor, and on the other side II. those without any or with a partial physical model. The first class comprises, as most common variant, inverse Finite Element Methods (FEM). These provide a fast path from local system reactions like mechanical strain derived from distributed sensors to the loads which cause them - in terms of strength level and location. This capability matches the description of a load monitoring system. Derivation and classification of damage introduced by these loads is a further issue asking for additional models describing e.

ISBN 978-3-527-33606-7

g. failure mechanisms and propagation. Re-introduction of structural change effected by such local damage to the structure requires another step, and a replacement of the original (inverse) structural model: This goes to show that the method has deficiencies when it comes to providing SHM systems with the necessary flexibility for real-time data evaluation which parallels damage evolution. Nevertheless, the method has its values where immediate model adaptation is less important and is thus object of ongoing research **[FRI01][VAZ06][DEN11][GHE11]**. The second class usually bases on classification or pattern recognition algorithms derived from supervised machine learning **[FAR13]**, discussed in Section **5.4**. A common widely used method is the correlation analysis of measured data resulting from an induced stimuli at run-time (system response) with data sets retrieved from an initial (first-hand) observation, which make it more difficult to select damage relevant features from the measurement results **[BOL09]**. These measurements base on signal correlation **[FAR13]** or waveform and spectral analysis, e.g. from vibration analysis **[CAR04][BAL06]**. Finite-element model-based simulation (FEM) can support and improve model-based sensor signal processing, for example, by identifying regions of interest which are damage-sensitive or by selecting appropriate regions to place sensors. Feature extraction using machine learning techniques with at least a model of the structure (but not necessarily of the sensor and sensor-material interaction) can be improved by FEM by substituting supervised learning under real conditions with a simulation-driven learning approach **[PAN11A]**.

Figure **5.27** shows the different information extraction classes representing different data processing levels and their relationship to the abstraction, as well as computational and data complexity.

Several different algorithms classes and levels of data processing are used in sensor networks, varying in computational and communication complexity, memory resources, and the kind of data they process. These are summarized in Fig. **5.28**.

Basically computation and data processing in sensor networks can be classified by:

- **Number of sensors**: Algorithms are performing computations based on one sensor value (local scope) or multiple sensor values (multiple-local or global scope),
- **Temporal computation model**: The computation is performed on-line or off-line in that sense that measuring data is either processed continuously at run-time or a limited set of data is only stored and processed after a measurement, with the first case requiring real-time capable data processing platforms,
- **Temporal processing model**: Processing of a computation is either strict sequentially (via a single process at time) or parallel processed (via multiple concurrent processes),

- **Spatial computation model**: Local (centralized) or global (distributed) computation is employed based on local or global distributed data, with either short- or long range data dependencies,
- **Spatial network model**: Computations can be performed in-situ (in-network) or outside of the sensor network (off-network),
- **Communication model**: Local computation is usually rooted in the shared memory model, whereas global computation uses message passing to exchange data.

Single sensor-based computations are mainly dealing with filtering, scaling, and calibration, whereas multiple-sensor based computations perform some kind of data fusion to improve the measuring quality and feature extraction producing higher level condensed information.

A deeper discussion of signal processing in sensor networks and actual and future challenges can be found in **[TRI12]**, pointing out the importance of distributed algorithms and data processing due to the inherent geometrical and distributed features of sensor networks, especially regarding material-embedded networks.

Algorithms can differ in their computational complexity significantly, which is usually increasing with increasing information level (abstraction). Therefore available computational resources of processing elements (concerning instruction complexity, average throughput, and data storage) of the sensor networks (e.g. the sensor nodes itself) limit the possible on-line/real-time capable implementations of higher level algorithms, such as machine learning, inverse numerical approaches, and complex pattern recognition. An off-line/on-line partitioning and co-design of algorithms is usually required to optimally satisfy resource and service constraints (hybrid data processing approach).

Scaling of algorithms (commonly used in information processing) to microchip level requires 1. simplification and 2. partitioning in programmable software tasks using microprocessors and non-programmable application-specific hardware tasks using finite-state machines and parallel multi-RTL architectures (for example **[BOS13B]**, with off-network but real-time machine learning and on-line autonomous distributed data processing in a sensor network performing sensor acquisition, calibration, pre-processing, and communication). Parallel data processing improves computational latency by preserving energy constraints and can be a prerequisite for real-time capable data processing platforms.

ISBN 978-3-527-33606-7

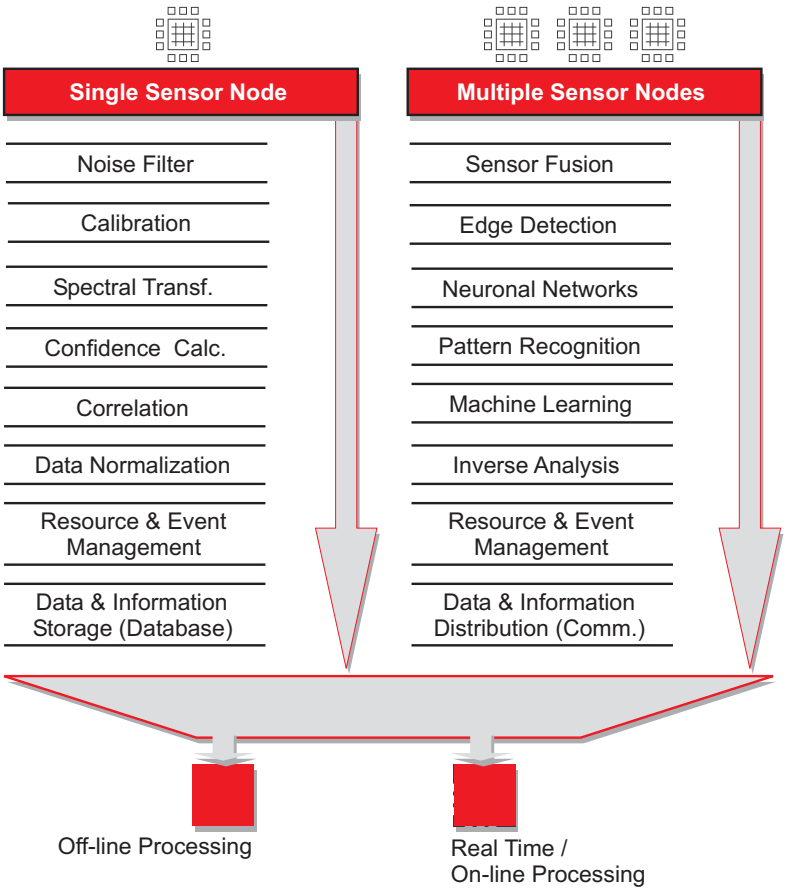


Fig. 5.28 *Different data processing algorithm classes used with single and multiple sensor nodes (i.e., deployed in distributed sensor networks)*

Many signal processing algorithms which can be partitioned in parallel functional networks can be parallelized, for example, communication (routing), data fusion **[MIT12]**, transformations (FFT), and many more.

Most current work of signal processing in sensor networks still implements sequential data processing on programmable machines (microcontrollers) for the sake of simplicity and flexibility of the design flow (rapid prototyping), thus failing to exploit concurrency and the advantages of application specific hardware design (examples can be found in **[ROS07][HED04][GHE10]**). The gap between the ongoing technological progress enabling increased miniaturization and node densities and the implementations itself become larger with those traditional serialized software based approaches. Traditional generic parallel microprocessor based architectures consume too much hardware (size & costs) and power resources which inhibit material integration.

For example, in many SHM applications correlation algorithms are used to identify signal deviations with external excitation as probing signal source (vibration analysis), thus providing an estimate of the change of state of the structure. Correlation functions generally have low computational complexity and offer the possibility for fine-grained parallelization, well suited for RTL data processing architectures.

Agent- and Machine Learning based approaches for SHM gain more importance in the past (see, for example, **[ZHA08]**).

5.2.3 Scientific Data Mining

Data mining is the process of computing meaningful information from raw data sets. More precisely, it reduces the size of the input compared to the output data vector. For example, a set of strain gauge sensors embedded in a technical structure is used to derive the health statements of the structure, which can be a simple Boolean value of the set {true, false} that is the output vector of the data processing. Now let us assume we have 100 sensors that are processed by autonomous sensor nodes capable of converting the sensor values with a 12 bit digital value. If time series evaluation is neglected, the input vector consists of 100 integer values (s_1, s_2, .., s_{100}), resulting in a 1200 bit vector, mapped finally on one bit! If history is relevant, the reduction in dimensionality is much higher. The information retrieval is quite easy if there is a model function defining a relationship between the sensor input data and the output information, and the process can be considered as analytical. But commonly this analytical function is missing, also if there is a mechanical model of the structure. In this case the data mining process is non trivial, and uses, for example, supervised machine learning methods or neural network approaches to approximate the mapping function. The sensor data can be multivariate, with data from sensors monitoring the same process or physical variable, and the input data can be at different spatial and temporal scales. Real world sensed data can contain missing values and the sensor data can be noisy and of low quality. Information retrieval must deal with this disturbed input data producing still correct and trustful data.

Data analysis is commonly an iterative and partial interactive process partitioned in multiple steps, summarized in Fig. **5.29** (details can be found in **[RAJ13]**). Each step depends on the kind of data to be processed and the relationship model between data and computed information. First of all the objects of interest must be localized or identified. In robotics using image analysis to identify geometric objects that are the specific objects to be found in an image. The features to be extracted can be related to a matching index (a probability to recognize a specific geometric shape), or the geometrical (center) position of an object to be tracked. Back propagation of data computed in one step to a previous step can be used for refinement of the analysis to improve analysis results and quality.

The analysis process is commonly not linear and discrete, mostly it will succeed to classify different analysis outcomes and objects. This is a classical separation problem, which is well known in machine learning techniques. For example, an image analysis should give

ISBN 978-3-527-33606-7

an answer with a significant evidence of the shape of an object detected in an image belonging to a small set of geometrical classes, i.e., {rectangular, circular, triangle}. An again we have a significant data reduction problem. Traditionally image analysis is related with high resolution camera data. But these approaches are not limited to camera data. Assume again the integration of a spatially distributed sensor network integrated in materials surface of a technical structure. The sensor data sampled by the sensor network can be compared and represented with a two-dimensional matrix like data sampled from a single image camera.

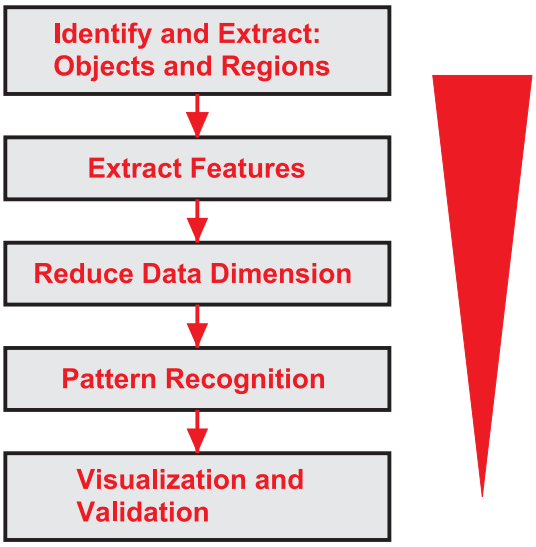


Fig. 5.29 *Levels of scientific data analysis used in data mining processes*

Feature Selection

Data reduction can be performed by using feature selection and transformation techniques **[RAJ13]**. For example, damage sensitive features should be extracted in a SHM application. Feature selection is used to discriminate a subset of features that are more relevant than others in determining the values of the variables composing the mapped information. These selected variables can be highly correlated with the output. Common functional feature selection techniques, beside machine learning approaches, are distances filters, chi-squared filter, stump filter, and quality estimation by the ReliefF method (Robnik-Sikonjaand Kononenko, 2003).

Correlation Techniques

Another commonly used analysis technique in SHM applications bases on temporal or spatial correlation analysis that identifies similarities between a pattern $\underline{pat}(x)$ and a tem-

plate *temp*(*x*), which can reduce a data vector to a scalar feature index value, e.g., the normalized peak value of the correlation function $\Gamma(\delta)=\langle pat(x)temp(x+\delta)\rangle$.

The auto-correlation function $\Gamma(\delta)=\langle XX\rangle$ describes the self-similarity of a continuous function *X* in relation to the lag δ of the function. The normalized correlation function has an output value interval [-1,1] (real output value). The cross-correlation function $\Gamma(\delta)=\langle XY\rangle$ computes the similarity of two continuous functions X and Y with a displacement lag δ, shown in Eq. **5.18**. The normalized cross-correlation function is scaled with the square root of the product of the auto-correlation functions of *X* and *Y* at lag zero.

*X** denotes the complex conjugate of the function *X*.

$$\begin{aligned}
\Gamma_{Y}(\delta) &= \left\langle Y^{*}(x)Y(x+\delta)\right\rangle = \lim_{X\to\infty} 1/X \int_{-X/2}^{X/2} Y^{*}(x)Y(x+\delta)dx \\
\Gamma_{YZ}(\delta) &= \left\langle Y(x)Z(x+\delta)\right\rangle = \lim_{X\to\infty} 1/X \int_{-X/2}^{X/2} Y^{*}(x)Z(x+\delta)dx \\
\Gamma_{YZ}^{norm}(\delta) &= \frac{\left\langle Y(x)Z(x+\delta)\right\rangle}{\sqrt{\left\langle Y(0)\right\rangle\left\langle T(0)\right\rangle}} \\
\overline{\Gamma}_{AB}(d) &= \left\langle A^{*}(i)B(i+d)\right\rangle = 1/X \sum_{i=-X/2}^{X/2} A^{*}(i)B(i+d)
\end{aligned} \tag{5.18}$$

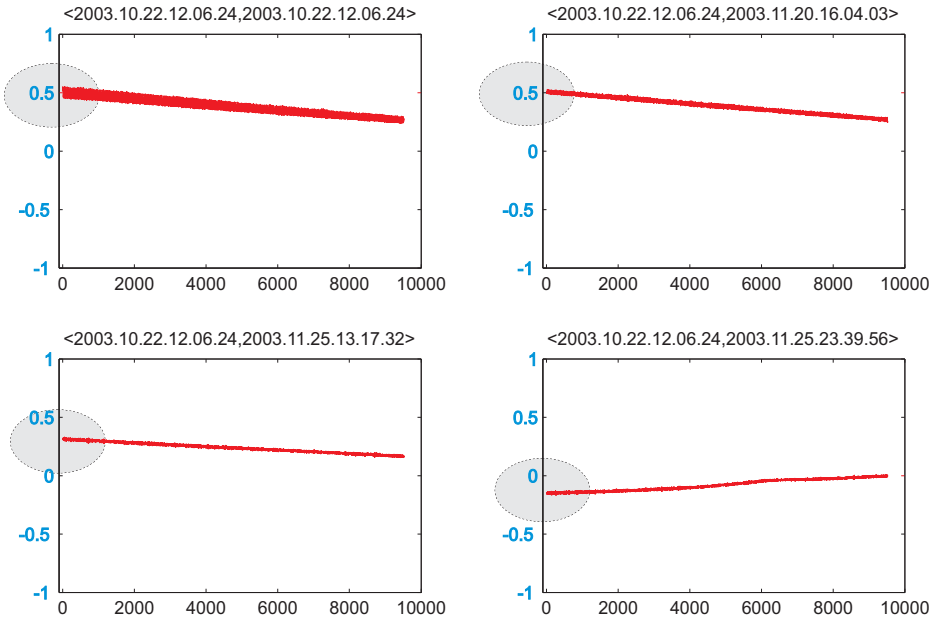
The discrete correlation function $\overline{\Gamma}(d)=\langle AB\rangle$ can be used for discrete finite data sets of functions *A* and *B*, e.g., $A=\{a_1,a_2,..\}$,$B=\{b_1,b_2,...\}$, respectively, with a discrete lag value *d*.

The following Ex. **5.1** shows the application of the discrete cross-correlation analysis to sensor data sampled from accelerometer (vibration) sensors attached to bearing housings.

In the first time of operation there is a significant correlation Γ(0) about 0.6 between the temporal resolved sensor data set (consisting of 2000 values sampled in one second), decreasing only slightly over the time. But at the end of the bearing lifetime, the correlation decreases significantly and finally vanishes, indicating a mechanical failure. The correlation analysis reduces therefore a vector of dimension 2000 to a scalar value (only Γ at a lag of zero is relevant for this SHM system).

ISBN 978-3-527-33606-7

Ex. 5.1 *Cross-correlation analysis of accelerometer sensor data from bearing operation comparing the time resolved sensor signal sample from a new bearing with sensor data from the aged bearing until a bearing failure occurs.*
[x: axis: displacement lag d, arb. units, at center position d=0!, y: axis: normalized crosscorr. function, arb. units, sensor data: NSF I/UCR Center for Intelligent Maintenance Systems]
(Top, Left): 3 minutes, (Top, Right): one month, (Bottom, Left): one month and three days, (Bottom, Right): ten hours later with bearing failure.



Correlation techniques can be also used for the spatial localization of sensors (proximity) towards intelligent and self-configuring distributed sensor networks by comparing the sensor data with sensor data from some sensors with a known sensor position using the cross-correlation approach **[HU13]**.

Feature Transformation

In contrast to feature selection algorithms the feature transformation maps a high dimensional input data vector on a lower dimensional output vector space. Prominent algorithms are Principal Component Analysis (PCA), that can be combined with DNA-based algorithms, Isomap (Tenenbaum et al.,2000), Locally Linear Embedding (LLE, Roweis and Saul, 2000), and Laplacian Eigenmaps (Belkin and Niyogi, 2000). The Laplacion Eigenmaps algorithms bases on the k-nearest neighbourhood approach, which is also applied in machine learning, resulting in a lower dimensional representation of the input data by giving the distances between data points.

5.2.4 Real-Time and Parallel Processing

Real-time signal processing systems must compute correct response of sensor data within a definite time-limit. That means the correctness of a computation in a real-time system depends on the correctness of the results and the time in which the results are produced. A violation of the specified time bounds either decreases system performance (soft real-time system) or results in malfunction (hard real-time system). The strictness of a deadline for a given computational task depends on the application to be performed.

There are different approaches to satisfy time bounds. First of all algorithms with constant run-time order are preferred, allowing run-time prediction at design-time, for example, using hash tables instead linear linked lists for resource management. Usually a computational task can be composed of several sub-tasks, which can be processed independently and in parallel, leading to a lower overall computation latency compared with the sequential processing case, supporting the satisfaction of run-time time bounds with safety windows. Parallel data processing can be classified in spatial parallelism and decomposition applied to one data set to be computed, and temporal parallelism and decomposition applied to a stream of data sets, aka. pipeline processing architectures. The latter one can improve processing element utilization and the data throughput of stream-based applications, but cannot improve the computation latency of one data set.

Parallel data processing requires the following steps **[TOK03]**:

- Decomposing computation into tasks on different levels of parallelism: sub-program, procedure, loop, expression, and bit level with decreasing granularity and increasing degree of parallelism from left to right,
- Assigning tasks to processes (independent computational units)
- Specification of inter-process communication (IPC) and synchronization (data access, exchange, and communication)
- Mapping processes to processing units (processors) for execution

It is usually not possible to map all processes of a computation to processing units with a 1:1 mapping due to resource constraints, resulting in simulated parallel processing by dynamically assigning processes to a processing unit, which executes several processes or part of them sequentially.

The performance of software executed on processors depends significantly on the instruction fetch-execute cycle and the load-execute-store model with the extensive use of register or memory variables by many high-level programming languages **[HAN06]**. Direct hardware synthesis of programs and algorithms can overcome these performance limitations.

The possible speedup that can be achieved by sub-task decomposition and parallel execution is limited by the fraction of serial execution r_s, which is considered as the non

ISBN 978-3-527-33606-7

parallelizable part of the program. This is addressed by Amdahl's law that expresses the limit of speedup by using N parallel processes in relation to the serial and parallel execution time fractions of a program, shown in Eq. **5.19** (with $r_s+r_p=1$).

$$Speedup(N) = \frac{1}{r_s + \frac{r_p}{N}} < N \tag{5.19}$$

Amdahl's law use the serial program as the reference. In contrast, Gustafson's law use the parallel program as the reference, and measures the serial fraction r_s' of the parallel program execution, and this is mainly a result from synchronization between parallel executing processes, shown in Eq. **5.20**.

$$Speedup(N) = r_s' + r_p' N < N \tag{5.20}$$

Amdahl's assumes a fixed total problem size, independent of the number of the processing units, whereby Gustafson's law assumes an increasing problem size with an increasing number of processing units, which can be more realistic in the context of Cloud Computing **[MOR12]**. That means that the parallel fraction increases with the number of processing units (and is not fixed).

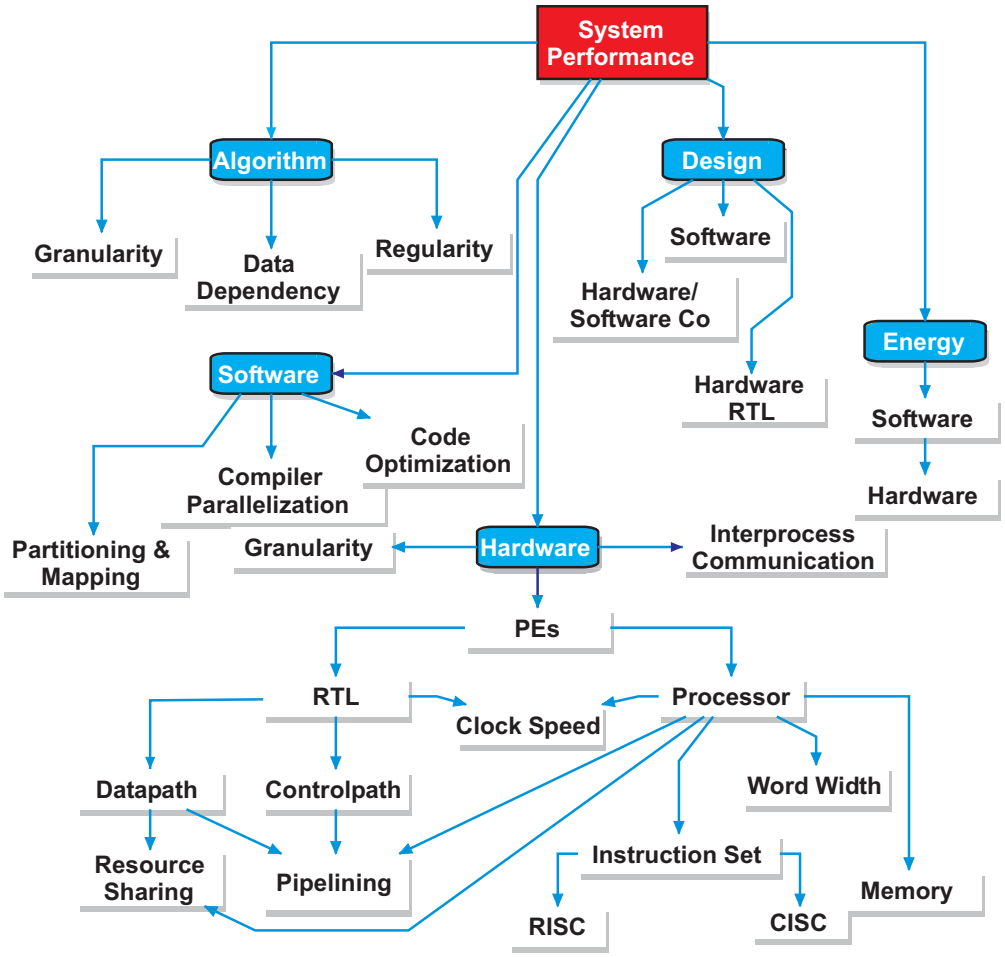


Fig. 5.30 *Main factors having impact on the design and the performance of data processing systems in sensor networks (adapted from [TOK03])*

The design of the data processing architecture used in material-integrated sensor networks requires the consideration and weighting of different impact factors, summarized in Fig. **5.30**, finding a merit for the satisfaction of resource requirements and computational time bounds.

Several sensor data fusion algorithms base on integral and sum terms (see **[MIT12]**, for example), which can be easily parallelized in the spatial dimension by partial or complete loop unrolling and data path parallelization, a powerful high-level optimization technique, which can be fully exploited by using RTL architectures, shown in Alg. **5.2**. Additionally, the application (calling) of functions can be parallelized by evaluating function argument expressions concurrently (due to inherent data independence). These techniques can help

ISBN 978-3-527-33606-7

to reduce the computation time and to meet time bounds significantly (or at least increasing the safety window).

Data pipeline processing can be applied to continuous data streams and provides a way to reduce the temporal spacing between single computation results of different data sets and improves the utilization of chained computational blocks (temporal parallelization), which can be fully exploited by using RTL architectures, shown in principle in Alg. **5.3**.

Alg. 5.2 *Parallelization rules for sum and product terms and function evaluation (||: sub-processes executed concurrently, I: partial expression computation or instruction sequence, →: operational sequence, $[e_1,..,e_n]\otimes/\oplus\rightarrow$:operational cluster block with n-ary operation)*

$$\frac{\Sigma_{i=a\ \text{to}\ b}\ I(i) \ \Rightarrow\ I(a) \rightarrow+ I(a+1) \rightarrow+ \ldots \rightarrow+ I(b)}{[I(a) \parallel I(a+1) \parallel \ldots \parallel I(b)] \rightarrow+}$$

$$\frac{\Pi_{i=a\ \text{to}\ b}\ I(i) \ \Rightarrow\ I(a) \rightarrow* I(a+1) \rightarrow* \ldots}{[I(a) \parallel I(a+1) \parallel \ldots \parallel I(b)]\otimes \rightarrow}$$

$$\frac{\Sigma_i\Sigma_{j=a\ \text{to}\ b}\ I(i,j) \Rightarrow I(a,a) \rightarrow+ I(a,a+1) \rightarrow+ \ldots}{[\Sigma_j\ I(i,a) \parallel \Sigma_j\ I(i,a+1) \parallel \ldots \parallel \Sigma_j\ I(i,b)]\oplus \rightarrow}$$

$$\frac{F(\varepsilon_1,\varepsilon_2,\varepsilon_3,\ldots\varepsilon_n) \ \Rightarrow\ v_1=\varepsilon_1 \rightarrow v_2=\varepsilon_2 \rightarrow \ldots \rightarrow F(v_1,v_2,..,v_n)}{[v_1=\varepsilon_1 \parallel v_2=\varepsilon_2 \parallel .. \parallel v_n=\varepsilon_n] \rightarrow F(v_1,v_2,\ldots v_n)}$$

Alg. 5.3 *Temporal parallelization rule with pipelining (simplified, s: input data, o: output data)*

$$\frac{\{..,s_i,..\} \Rightarrow\ F(G(H(..(X(s_i)))) \ \Rightarrow\ \{..,o_i,..\}}{\begin{array}{l}\{..,s_{i-m},s_{i-m+1},..,s_i,..\} \Rightarrow \\ \parallel[\ X(s_i) \rightarrow \ldots \rightarrow H(s_{i-m+2}) \rightarrow\ G(s_{i-m+1}) \rightarrow F(s_{i-m})] \Rightarrow \\ \{..,o_{i-m},o_{i-m+1},..,o_i,..\}\end{array}}$$

One major feature of parallel and distributed algorithms is their data dependency, which can be classified in local short-distance and global long-distance dependency regarding tasks or partitions of a program, as shown in Fig. **5.31**. Inter-task data dependency causes communication during the parallel or distributed data processing, which can significantly lower the available speed-up. Commonly, global data dependency causes the highest communication overhead. Communication is usually a synchronization, too, and therefore increases the sequential part of a parallel program.

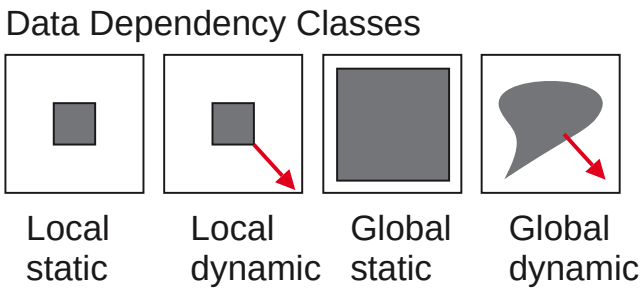


Fig. 5.31 *Different data dependency classes of algorithms.*

Register-Transfer Level Architectures

Spatial parallelism leads to an increase of resources by replication, whereby temporal parallelism leads to an increase of the utilization of a sequentially composed system. This pipelining can be very versatile in data flow computations, allowing the design and the proper control of pipelines at design time (e.g., RTL architectures with data path pipelining). In contrast control flow based computations like machine programs executed on microprocessors make the design of pipelines difficult with a lower ratio of real speed up to resource costs due to the non-predictability of branches in the control flow and the expensive reset logic.

Application specific designed digital logic circuits with RTL architectures allow the implementation of data latency, throughput, and resource optimized data processing systems on microchip level. The principle structure of RT architectures is shown in Fig. **5.32**. Basically combinatorial logics performs functional computations $f(o_1,o_2,..)$ of data stored in registers. The data-path that performs sequential data processing consists of multiple levels with alternating registers and functional blocks between these registers levels. The input operands o_i of the n-th level function f_n are the input data from registers of level n, and the results of the computation is transferred to registers of level n+1. This data path architecture offers parallel data processing on expression level and temporal pipelining of different datasets processed in different levels (stages) of the pipeline. Combinatorial logic introduces disadvantages concerned with metastability and path delays leading to increased switching activity and power consumption. In a synchronous system with a central master clock the highest possible clock frequency is determined by the longest path delay in a combinatorial block. Refactoring of the data path in small functional blocks weaved between register levels increases the overall system performance. Registers are barriers for (invalid) metastable digital logic levels between combinatorial blocks.

ISBN 978-3-527-33606-7

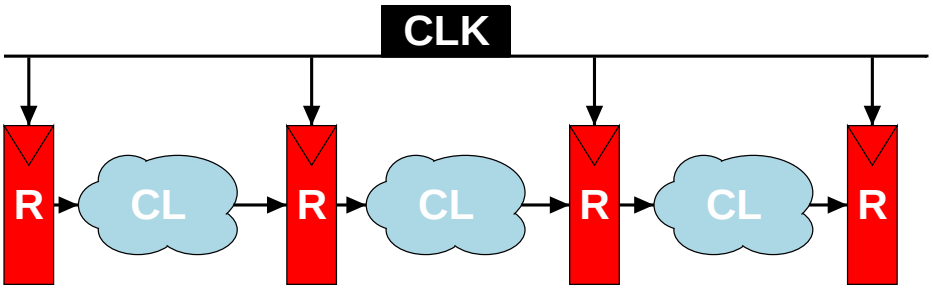


Fig. 5.32 *Principle data processing with register-transfer architectures (synchronous system) [R: Register, CL: Combinatorial Logic, CLK: Common clock signal)*

The data processing in RTL architectures is performed sequentially and controlled by a Finite State Machine (FSM) meeting todays still common formulation of algorithms and communication in a sequential order, shown in Fig. **5.33**. The FSM partitioned the computation in multiple steps, defining the control path, with states that activates specific register stages of the data path part of the RTL system. The data path is usually composed of the multiple sub-paths and consists of functional blocks, data path selectors (multi- and demultiplexer), registers, and addressable cell memories (RAM). Path selectors are used for switching input and output ports of shared resources like functional blocks (adders, multipliers,...) The data path is bidirectionally connected with the control machine and the outside world, interfacing, for example, sensor acquisition or communication devices.

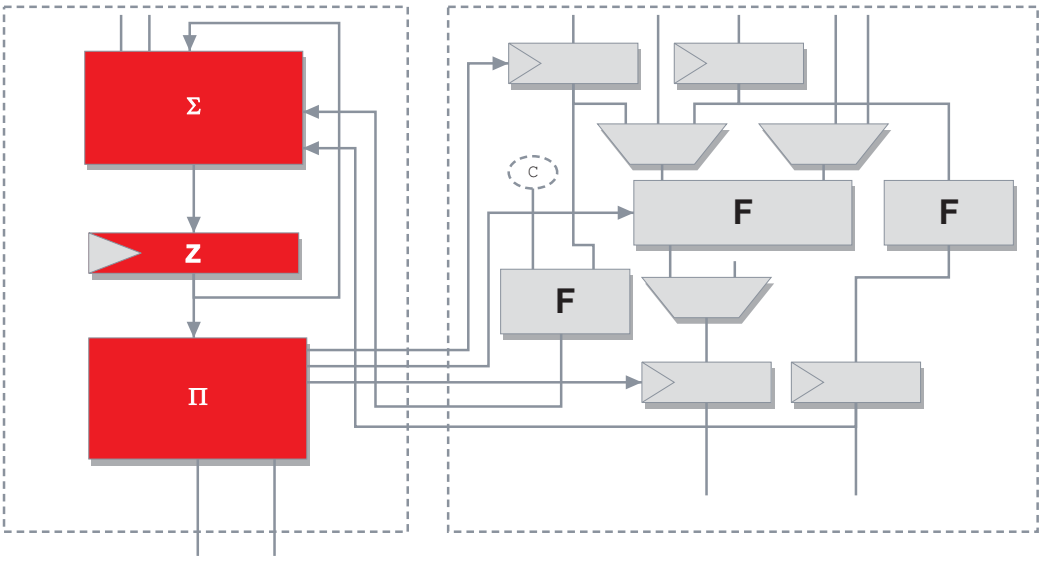


Fig. 5.33 *RTL Data Processing Architecture (Left) Finite State Machine Controller (Right) Data path and Computation*

 ISBN 978-3-527-33606-7

The Finite State Machine controller consists of a state registers storing the encoded state representation (i.e., binary coding), a state-transition network Σ computing the next state, and an output data generator Π, providing the control signals for the data path.

Virtual Machine Architectures

Virtual machine concepts are increasingly employed in sensor networks supporting abstraction and virtualization (discussed in **[SMI05]**) of the technological platform and offering advanced flexibility and programming features, including on-the-fly compilation of program code at run-time **[MUE07]**. A Virtual machine not only offers an abstraction layer of the processing system, it offers the virtualization of resources, like storage, sensor, or network resources, too. They enable the distribution of programs and migration of mobile processes in an architecture-neutral format, which can easily be interpreted or compiled **[GRE08]**, usually stored in and directly executed from byte code formats. There are two different classes of virtual machines, which are capable to execute byte code: Register-based and stack-based processing machines. Register-based machines commonly perform computation by modifying local register or shared memory using multiple-operand instructions, whereby stack-based machines operate on one or multiple stacks using zero-operand instructions, which access or modify few top elements of the stack. Zero-operand instruction formats offer an in-system programmable code morphing capability, the ability of program code to modify itself (in-place or out-of-place) **[BOS12B]**, which can be used, for example, for agent behaviour modification at run-time. Stack-based machine designs, originally closely related to the *FORTH* programming language invented by Chuck Moore, enable advanced resource and power optimizations **[CHU11B]**. Due to the functional paradigm stack-based machines are well suited for the implementation of massive parallel systems **[CHU11A]**. But common programming languages base on a random access memory model, which does not match the computation model of a stack-based machine, requiring the transformation of register-based to stack-based computation (examples of efficient transformation rules are explained in **[PAR11]**).

Virtual machines (VM) can be executed in restricted user space (non-privileged) or in unrestricted system space (privileged mode), shown in Fig. **5.34**. Non-privileged applications have restricted access to system and hardware resources. Commonly software is executed directly on hardware (RTL architectures and simple embedded microcontroller systems) in the unrestricted execution space, or on the top of an operating system (OS) in restricted mode, e.g., with additional memory and code protection. A virtual machine, executing an application, can be directly executed on hardware or on top of an operating system, too. In server applications a VM can be used ot execute a guest OS that executes applications. Virtual machine monitors can be used to translate between the privileged and the non-privileged execution space. In distributed sensor network applications there are commonly embedded systems with low resources that prohibits the use of operating systems, delegating the abstraction of hardware and the system services to the application

software. Virtual machines can provide a meaningful middle layer incorporating operating system services required by application software and providing the necessary hardware abstraction, for example, network communication, sensor representation, and memory management, to make the application independent from the hardware layer. Furthermore, virtual machines can provide security and robustness by monitoring and automatic memory management based on garbage collection. There are several VM architectures, mostly based on byte-code execution, which are suitable for low-resource embedded systems. FORTH stack-based machines are one example, another is the OCaML byte-code interpreter, which was successfully implemented on a low-resource 8-bit microcontroller **[VAU11]**, or the Agent Forth VM **[BOS15A]**, suitable for pure RTL microchip hardware implementations. In contrast to stack-based architectures, the tinyRef architecture **[MAR09]** promotes a register-base VM for wireless sensor nodes, arguing with lower code size and increased execution speed (the program execution costs), but which cannot be hold generally as shown in **[BOS15A]**.

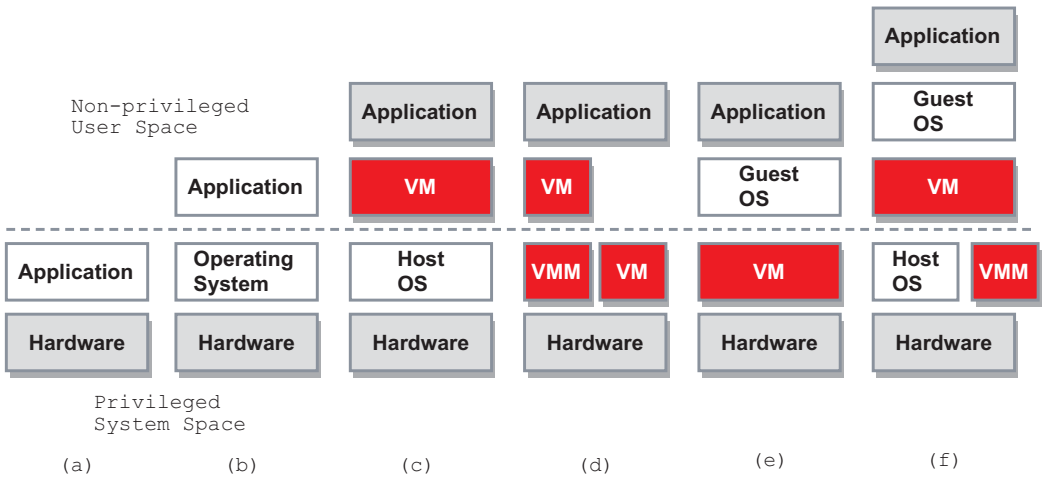


Fig. 5.34 *Different Virtual Machine (VM) Architectures with and without a Virtual Machine Monitor translating between system and non-privileged user processing space. (a): Direct applications executed by hardware, (b): with operating system (OS), (c): with virtual machine executed in user space, (d): with VM in system and user space bridged with a VMM, (e): with a guest OS, and a guest OS executed on a user space VM.*

5.3 The Known World: Model-based Computing and Inverse Numeric [Lechleiter, Bosse]

In this chapter we consider parameter identification problems and discuss several deterministic techniques for the extraction of information on searched-for parameters from sensor data. These techniques will rely on the one hand on some physical model for the data acquisition process and on the other hand on deterministic mathematical algorithms to stably compute searched-for quantities from measured sensor data in the context of ill-conditioned or under-determined problems. In engineering, physics, or other sciences, the process of inverting measured data in order to approximate time- and/or space-dependent model parameters typically lacks stability whenever quantities of interest can only be accessed indirectly via remote measurements or via measurements of other physical quantities. A prominent example in this class of problems is the approximation or characterization of material properties from boundary measurements of the material's behaviour in non-destructive testing or structural health monitoring.

If only remote or indirect data is available, there typically exist quite different parameters that lead to rather similar sensor measurements such that their stable and accurate inversion becomes a challenging problem. In particular, unavoidable noise in the sensor measurements implies that any inversion scheme has to be stabilized in some suitable way, a concept which is also known as regularization in the terminology of applied mathematics. Once a computable physical model linking model parameters with sensor measurements is at hand, regularization theory offers inversion algorithms that stably compute model parameters from noisy sensor data by, roughly speaking, balancing the amount of information extracted from the noisy data with the level of stability of the inversion process.

In this chapter, we aim to highlight several parts of linear and non-linear regularization theory adapted to applied scientists familiar with concepts from basic calculus; consequently, we simplify settings and results to ease access to the material. Textbooks and monographs on regularization theory that go far beyond what can be sketched in the overview on the following pages include [EHN96, Kir96, MS12] and [SKHK12, KNS08], respectively.

5.3.1 Physical Models in Parameter Identification

Taking the viewpoint of an applied scientist, any parameter identification problem searches for an element in some parameter set that is able to explain the measured data. The fundamental advantage of a computable physical model for some data acquisition process is that the implicit link between model parameters and sensor measurements becomes predictable, typically at the expense of a numerical simulation. If measured sensor data are available, such a model offers, at least in principle, the possibility to test for a variety of parameters whether numerically simulated sensor measurements for test parameters match

ISBN 978-3-527-33606-7

these measurements. In other words, a physical model offers the opportunity to check whether a given parameter explains the sensor measurements under the chosen physical model.

Taking a more abstract viewpoint, a physical model for some data measurement process aiming to characterize unknown parameters provides a mapping Φ from the set of admissible parameters $\mathcal{P}$ into the space of sensor measurements $\mathcal{S}$,

$$\Phi: \mathcal{P} \to \mathcal{S},$$

such that we will in the following identify Φ with the corresponding model. Disregarding possibly time-dependent measurements, $\mathcal{S}$ can be most often simply be identified with some Euclidean space $\mathbb{R}^N$ of dimension N, corresponding to N sensor measurements. To make this text easier accessible for applied scientists, we are always going to take this viewpoint in the sequel and set $\mathcal{S} = \mathbb{R}^N$ for some $N \in \mathbb{N}$.

The above-described parameter identification task can then be – naively – rephrased as follows: Given sensor measurements $s \in \mathcal{S}$, determine suitable parameters $p \in \mathcal{P}$ such that $\Phi(p) = s$. The model Φ is usually called the forward operator within the mathematical community; the parameter identification problem hence consists in stably inverting this forward operator, in order to find parameters explaining sensor data.

Example 1 (Inverse problems in linear elasticity)
We illustrate the above concept with an example from parameter identification in linear elasticity. In structural health monitoring, one is interested in computing material parameters of a body $\Omega \subset \mathbb{R}^3$ from measurements of its material behaviour. Let us for simplicity assume that the material behaviour of Ω is linearly elastic, such that there is a stiffness tensor $C: \Omega \to \mathbb{R}^{3\times3\times3\times3}$ characterising the material behaviour of Ω. More precisely, for each $x \in \Omega$, $C(x)$ is fourth-order tensor that linearly links the Cauchy stress tensor $\sigma: \Omega \to \mathbb{R}^{3\times3}_{sym}$ with the infinitesimal strain tensor $\varepsilon: \Omega \to \mathbb{R}^{3\times3}_{sym}$ via Hooke's law,

$$\sigma(x) = C(x): \varepsilon(x), \qquad x \in \Omega,$$

that is, $\sigma_{ij}(x) = \sum_{k,l=1}^{3} C_{ijkl}\varepsilon_{k,l}$. Without going into details, the stiffness tensor C is a positive definite linear mapping on symmetric 3×3 matrices, which possesses in total 21 degrees of freedom at each point x in Ω, due to symmetries (see [Cia88, Gou13]). The infinitesimal strain tensor is explicitly determined by the deformation field $u: \Omega \to \mathbb{R}^3$,

$$\varepsilon(x) = \varepsilon(u(x)) = \frac{1}{2}[\nabla u(x) + (\nabla u(x))^T], \qquad x \in \Omega. \tag{5.21}$$

such that the continuum equation yields the following governing equation for the deformation field due to some applied body force $f: \Omega \to \mathbb{R}^3$,

$$div(C: \varepsilon(u(x))) = f(x), \qquad x \in \Omega. \tag{5.22}$$

Suitable boundary conditions include displacement boundary conditions for the restriction of u to some subset Γ_1 of the boundary $\partial\Omega$,

$$u(x) = g_1(x), \qquad x \in \Gamma_1 \subset \partial\Omega, \tag{5.23}$$

stress boundary conditions for the traction $\sigma\nu = (C{:}\,\varepsilon(u))\nu$ (where ν denotes the exterior unit normal to $\partial\Omega$),

$$\sigma(x)\nu(x) = g_2(x), \qquad x \in \Gamma_2 = \partial\Omega\backslash\Gamma_1, \tag{5.24}$$

or linear combinations between these two types. If Γ_1 is a non-empty subset of $\partial\Omega$, and if $g_{1,2}$ are sufficiently regular, then there exists a unique deformation field solving the corresponding boundary value problem. For simplicity, we assume in the following that $g_{1,2} = 0$.

In principle, inversion of C from sensor measurements can be based on surface strain measurements on Γ_1 or on surface deformation measurements on Γ_2. Components of the surface strain $\varepsilon(u(x))$ at boundary points $x \in \partial\Omega$ can be measured by, e.g., applied strain gauges or fibre Bragg gratings, whilst surface deformation can be measured optically using interferometry. Neglecting that these measurements should be adequately modelled as well, the forward operator Φ then maps parameters in the set $\mathcal{P}$ consisting of positive-definite and symmetric fourth-order tensors to a finite number N of sensor measurements of surface strain and/or surface deformation of the field u from (5.22). The inversion problem for the material parameter C can then be formulated as follows: Determine the fourth-order symmetric and positive-definite tensor C from spatially distributed sensor measurements $s \in \mathcal{S} = \mathbb{R}^N$ of surface strains and/or surface deformation.

It is important to note that linearity of the differential equation in (5.22) means that the deformation field u as well as its surface strains and surface deformation depend linearly on the applied body force f. However, as u clearly also depends on the stiffness tensor C, the application of C to $\varepsilon(u)$ in (5.22) implies that u does *not* depend linearly on C, such that the forward operator Φ is non-linear. This fact makes the introduced inversion problem challenging, even for stiffness tensors that are spatially constant or piecewise constant.

As more sensor measurements yield a larger amount of (implicit) information on the stiffness tensor, the stability of any inversion algorithm is enhanced by collecting more data either by relying on more sensors or by applying several forces f to the body. (Of course, the analogous inversion problem can be posed and the same properties hold if the deformation field is excited by a deformation $g_1 \neq 0$ or surface traction $g_2 \neq 0$.)

A somewhat different identification problem arises if the material coefficients of Ω are known but forces acting on Ω are the searched-for quantity; thus, whilst the stiffness tensor C in (5.22) is known, the volume force f is the searched-for parameter. This setting is often met in load monitoring where one is interested in identifying loads on structures from sensor information on the material behaviour under load. The forward mapping Φ

ISBN 978-3-527-33606-7

hence maps forces f in some function space $\mathcal{P}$ on Ω (e,g., in the space of piecewise linear functions on some triangular or tetrahedral mesh decomposing Ω) to sensor measurements s_m of the surface deformation and/or surface strain caused by the deformation field due to f. As discussed above, the measurements s_m depend linearly on the deformation field u, which itself depends linearly on f, such that $\Phi: f \mapsto s_\mathrm{m}$ is a linear mapping and the inversion problem to find a force f such that $\Phi(f) = s_m$ for given s_m is linear, too. We are going to see in the sequel that linear identification problems are somewhat easier to tackle compared to non-linear ones; in particular, their analysis is significantly simpler.

5.3.2 Noisy Data due to Sensor and Modeling Errors

Unfortunately, real-world sensor measurements are never ideal since the usual sensor noise together with various sensor errors perturb the sensor signal ahead of any data processing (sensor errors include, e.g., digitalization, sensitivity or drift errors as well as possible environmental influences as temperature or moisture). Similarly, no physical model can exactly reproduce real-world phenomena, such that modeling errors are further error sources. Thus, any measured signal comprises a certain level of noise $\delta > 0$ coming from the two mentioned sources. Notationally, we take this noise level into account by writing s_m^δ for noisy data in $\mathcal{S}$, while $s_\mathrm{m} \in \mathcal{S}$ from now on denotes noise-free data according to the chosen model Φ; for $s_\mathrm{m} \in \mathcal{S}$ there hence exists $p \in \mathcal{P}$ such that $\Phi(p) = s_\mathrm{m}$. The absolute noise level δ is, by definition, the magnitude of the (unknown) difference of s_m and s_m^δ in $\mathcal{S} = \mathbb{R}^N$,

$$\delta = \| s_\mathrm{m}^\delta - s_\mathrm{m} \|_{\mathbb{R}^N} = \left(\sum_{j=1}^N |s_{\mathrm{m},i}^\delta - s_{\mathrm{m},i}|^2\right)^{\frac{1}{2}} \quad \text{for } s_\mathrm{m}^\delta = (s_{\mathrm{m},i}^\delta)_{i=1}^N,\ s_\mathrm{m} = (s_{\mathrm{m},i}^\delta)_{i=1}^N.$$

We will later on consider noise levels $\delta > 0$ that tend to zero and denote this as $\delta \to 0$; in this case, the definition of δ in the last equation implies that $s_\mathrm{m}^\delta \to s_\mathrm{m}$ in $\mathbb{R}^N$.

Note that noisy data s_m^δ does, in general, not belong to the range of Φ, roughly speaking since noise usually does not feature smoothness; even more, it is rarely the case that one finds some model parameter p that satisfies the equation $\Phi(p) = s_\mathrm{m}^\delta$ exactly. For this reason, it is *not* advisable to attempt to solve the latter equation for given sensor data s_m^δ exactly, as any resulting solution would be required to explain the noisy data, despite we are *not* interested in explaining the noise $s_\mathrm{m}^\delta - s_\mathrm{m}$.

Taking a more theoretic viewpoint, whenever data is gained by remote or indirect measurements, the mapping Φ usually fails to possess a continuous inverse Φ^{-1} (if ever this inverse mapping exists at all). Even if Φ^{-1} exists, its discontinuity implies that merely computing $\Phi^{-1}(s_\mathrm{m}^\delta)$ fails to provide sense-full parameter reconstructions in practice, as $\Phi^{-1}(s_\mathrm{m}^\delta)$ is far from $\Phi^{-1}(s_\mathrm{m})$ even for tiny noise levels δ. Instead, one should attempt to stabilize the inversion process by solve the operator equation approximately up to the noise level δ of the sensor data. We are going to illustrate several numerical techniques to do so in the subsequent sections.

Before, let us demonstrate by a simple linear problem in a finite-dimensional setting, motivated by an inverse problem in heat conduction, what can go wrong if one tries to exactly solve operator equations $\Phi(p) = s_{\mathrm{m}}^{\delta}$ for parameter identification problems.

Example 2 (Inverse heat conduction)
Assume we are given sensor measurements $s_m^{\delta} \in \mathbb{R}^N$ of the temperature of a thin, homogeneous bar of length one at some time $t_0 \gg 0$; the inversion problem under consideration is to compute the initial temperature of the bar at time $t = 0$.

As a simplistic physical model for heat conduction in the bar, we choose the one-dimensional heat equation with constant thermal diffusivity $\alpha > 0$ for the temperature u,

$$\frac{\partial u}{\partial t} - \alpha \frac{\partial^2 u}{\partial x^2} = 0 \quad \text{for } x \in [0,1], t > 0,$$

subject to initial and (isolating) boundary conditions

$$u(0,x) = g(x) \quad \text{for} x \in [0,1] \qquad \text{and} \qquad \frac{\partial u}{\partial x}(t,0) = \frac{\partial u}{\partial x}(t,1) = 0 \quad \text{for } t > 0.$$

Solutions to this initial boundary-value problem are of the form

$$u(t,x) = \sum_{n \in \mathbb{N}_0} \hat{u}_n \cos(\pi n x) e^{-\alpha \pi^2 n^2 t} \quad \text{for } x \in [0,1], t > 0, \quad \text{for coefficients } \hat{u}_n \in \mathbb{R}. \tag{5.25}$$

For N sensor positions $0 \leq x_1 < x_2 < \cdots < x_N \leq 1$, the forward operator Φ maps the initial value g, a function of x, to the vector $s_{\mathrm{m}} = (u(t_0,x_j))_{j=1}^{N} \in \mathbb{R}^N$ containing point values of the temperature at time $t_0 > 0$. Thus, s_{m}^{δ} contains noisy sensor measurements of the temperature at time $t_0 > 0$ at the sensor positions x_j, such that $\delta^2 = \sum_{j=1}^{N} |s_{\mathrm{m},j}^{\delta} - u(t_0,x_j)|^2$. The operator equation to be tackled hence reads $\Phi(g) = s_{\mathrm{m}}^{\delta}$ for given measurements s_{m}^{δ} and a searched-for initial temperature g. A straightforward solution to the operator equation $\Phi(g) = s_{\mathrm{m}}^{\delta}$ exists once we compute a trigonometric interpolation

$$u_{t_0,N}^{\delta}(x) = \sum_{n=0}^{N-1} \hat{u}_n^{\delta} \cos(\pi n x) \quad \text{such that } u_{t_0,N}^{\delta}(x_j) = s_{\mathrm{m},j}^{\delta} \text{ for } 1 \leq j \leq N. \tag{5.26}$$

This interpolation is particularly easy in case that $x_j = (2j-1)/(2N)$ for $j = 1, \ldots, N$, since the discrete cosine transform of the sensor values then yields the coefficients $\hat{u}_n^{\delta}$,

$$\hat{u}_0^{\delta} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s_{\mathrm{m},k}^{\delta}, \qquad \hat{u}_n^{\delta} = \frac{\sqrt{2}}{\sqrt{N}} \sum_{k=0}^{N-1} s_{\mathrm{m},k}^{\delta} \cos(\pi x_k n) \quad \text{for} 1 \leq n \leq N-1.$$

For simplicity, we hence assume from now on that $x_j = (2j-1)/(2N)$. Then (6) takes the explicit form

ISBN 978-3-527-33606-7

$$u^{\delta}_{t_0,N}(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} + \frac{\sqrt{2}}{\sqrt{N}} \sum_{n=1}^{N-1} \sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} \cos(\pi x_k n) \cos(\pi n x), \quad x \in [0,1]. \tag{5.27}$$

Matching the expression of $u^{\delta}_{t_0,N}$ at time $t = t_0$ in (5.27) with the solution u in (5.25) shows that the coefficients $\hat{u}_n$ of u from (5.27) are given by

$$\hat{u}_0 = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k}, \quad \hat{u}_n = \frac{\sqrt{2}}{\sqrt{N}} \left(\sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} \cos(\pi x_k n) \right) \exp(\alpha \pi^2 n^2 t_0), \quad 1 \leq n \leq N-1,$$

and $\hat{u}_n = 0$ for $n \geq N$. The solution to the heat equation that equals $u^{\delta}_{t_0,N}$ at time $t = t_0$ thus equals

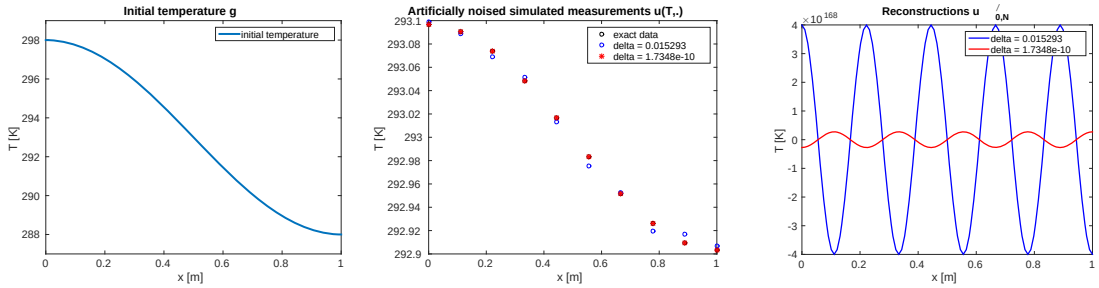
$$u^{\delta}_N(t,x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} + \frac{\sqrt{2}}{\sqrt{N}} \sum_{n=0}^{N-1} \left(\sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} \cos(\pi x_k n) \right) \cos(\pi n x) e^{\alpha \pi^2 n^2 (t_0 - t)},$$

and takes initial values

$$u^{\delta}_N(0,x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} + \frac{\sqrt{2}}{\sqrt{N}} \sum_{n=0}^{N-1} \left(\sum_{k=0}^{N-1} s^{\delta}_{\mathrm{m},k} \cos(\pi x_k n) \right) \cos(\pi n x) e^{\alpha \pi^2 n^2 t_0} \tag{5.28}$$

for $x \in [0,1]$. Consequently, $\Phi(u^{\delta}_N(0,x)) = (u^{\delta}_N(t_0,x))_{j=1}^{N} = s^{\delta}_{\mathrm{m}}$, that is, $u^{\delta}_{0,N}$ is an exact solution to the operator equation we are considering.

However, for all but the constant term in (5.28) the noise in s^{δ}_{m} into u^{δ}_n is multiplied with the exponentially growing factors $\exp(\alpha \pi^2 n^2 t_0)$, such that any attempt to find reasonable initial temperatures will completely fail due to a blow-off of the exponential terms in (5.28), which is also clearly visible in Figure **5.35**.



(a) (b) (c)

Fig. 5.35 *Fig. 1: Inverse heat conduction (parameters:* $N = 10$, $T = 4$, $\alpha = 0.2$, $g(x) = 293 + 5\cos(\pi x)\exp(-\alpha\pi^2 T)$*). (a) Plot of* g *(b)* s_{m} *and* s_{m}^{δ} *for* $\delta = 10^{-2}$ *and* $\delta = 10^{-10}$ *(c)* $u_{0,N}^{\delta}$ *for* $\delta = 10^{-2}$ *and* $\delta = 10^{-10}$ *Obviously, the explicit formula for the reconstruction* $u_{0,N}^{\delta}$ *is worthless as its magnitude has blown up due to instability caused by the exponential factor in (5.28) that roughly equals* $\exp(7.896\, n^2) \approx 2686^{n^2}$.

5.3.3 *Coping with Noisy Data: Tikhonov Regularisation and Parameter Choice Rules*

Since parameter identification problems typically lack stability, one should refrain from attempting to solve the operator equation $\Phi(p) = s_{\mathrm{m}}^{\delta}$ exactly. As Example 2 shows, unavoidable measurement errors lead to completely wrong solutions that suffer in particular from a blow-up in magnitude. A first idea is hence not to try to find a parameter $p \in \mathcal{P}$ such that $\Phi(p)$ equals s_{m}^{δ}, but to relax this criterion and merely attempt to find $p \in \mathcal{P}$ such that on the one hand the discrepancy $\| \Phi(p) - s_{\mathrm{m}}^{\delta} \|_{\mathbb{R}^N}$ is small but on the other hand the magnitude of the reconstructed parameter p is not too large.

This idea requires to formalise the concept of the "magnitude" of a parameter mathematically (having, e.g., the stiffness tensor $C\colon \Omega \to \mathbb{R}^{3\times3\times3\times3}$, the force $f\colon \Omega \to \mathbb{R}^3$, or the initial temperature $g\colon [0,1] \to \mathbb{R}$ as parameters in Examples 1 and 2 in mind). Such a formalisation is for instance achieved by basic functional analytic concepts such as infinite-dimensional Banach- or Hilbert spaces, and norms and operators on such spaces; this mathematical theory is introduced in detail in standard textbooks as, e.g., [Rud91, Aub00]. To avoid the need for a detailed exposition of relevant infinite-dimensional theory, we opt to restrict ourselves to finite-dimensional parameter spaces X with scalar product $(\cdot,\cdot)_X$ and norm $\|\cdot\|_X = (\cdot,\cdot)_X^{1/2}$, such that $\mathcal{P} \subset X$ is subset of a finite-dimensional vector space. This situation occurs for instance automatically after discretization of an infinite-dimensional problem and eases the presentation considerably. In the simplest case, $X = \mathbb{R}^L$ for $L \in \mathbb{N}$, such that $(p_1, p_2)_X = p_1 \cdot p_2$ is the usual scalar product of two vectors. If X is a function space spanned by finitely many basis functions $\{b_1, \dots, b_L\}$, then each $p = \sum_{j=1}^{L} v_j b_j \in X$ is represented by a unique vector $v = v_p \in \mathbb{R}^L$ containing the coeffi-

ISBN 978-3-527-33606-7

cients v_j of p for the basis functions b_j; a scalar product on X can be defined, e.g., via the standard scalar product on $\mathbb{R}^L$ of the coefficient vectors, $(p, p')_X = v_p \cdot v_{p'}$.

Tikhonov Regularization

Coming back to our above idea to find parameters $p \in \mathcal{P}$ that balance the discrepancy $\| \Phi(p) - s_{\mathrm{m}}^{\delta} \|_{\mathbb{R}^N}$ with the magnitude of p, we exploit the X-norm $\|\cdot\|_X$ to measure that magnitude and rely on a relaxation parameter $\alpha > 0$ to define suitable parameters as solutions to the minimization problem

$$\min_{p \in \mathcal{P}} [\| \Phi(p) - s_{\mathrm{m}}^{\delta} \|_{\mathbb{R}^N}^2 + \alpha \| p \|_X^2] .$$

As it is rather frequent that not all elements in X belong to the set of admissible parameters $\mathcal{P} \subset X$ due to, e.g., sign conditions coming from the non-negativity of several physical quantities whenever those are searched-for parameters, the minimum in the last equation is indeed searched for over $\mathcal{P}$ instead of X. The penalty term $\|\cdot\|_X^2$ can further be exploited to incorporate a-priori knowledge on the searched-for parameter p by considering the functional

$$\mathcal{J}_{\alpha, s_{\mathrm{m}}^{\delta}}(p) = \| W\big(\Phi(p) - s_{\mathrm{m}}^{\delta}\big) \|_{\mathbb{R}^N}^2 + \alpha \| D(p - p^*) \|_Z^2 , \quad \text{for } p \in \mathcal{P}, \tag{5.29}$$

some invertible $N \times N$ matrix W, an element $p^* \in \mathcal{P}$ (or in X) and some injective linear mapping D from the finite-dimensional space X into another finite-dimensional vector space Y. The matrix W modifies the norm on $\mathbb{R}^N$ and is typically chosen as inverse of the Cholesky factor of the covariance matrix of s_{m}^{δ} (if a statistical distribution of s_{m}^{δ} is known), see [ZH90]; the mapping D is often chosen as a (discretized) differential operator that forces (piecewise) smoothness of the minimizer. We denote any minimizer of the latter functional on $\mathcal{P}$ by p_{α}^{δ},

$$p_{\alpha}^{\delta} = argmin_{p \in \mathcal{P}} \mathcal{J}_{\alpha, s_{\mathrm{m}}^{\delta}}(p). \tag{5.30}$$

Note that a minimiser p_{α}^{δ} of the functional $\mathcal{J}_{\alpha, s_{\mathrm{m}}^{\delta}}$ is not necessarily unique for a non-linear model Φ, such that we cannot speak of *the* minimizer to (5.30).

The functional $\mathcal{J}_{\alpha, s_{\mathrm{m}}^{\delta}}$ from (5.30) is the so-called Tikhonov functional with regularization parameter α, and the technique to determine stabilized solutions to $\Phi(p) = s_{\mathrm{m}}^{\delta}$ by (5.30) is called Tikhonov regularization. (For linear mappings Φ, we later on consider a simple way to compute the minimizer by solving a linear equation.) Obviously, the choice of α crucially determines properties of resulting minimizers p_{α}^{δ}. Thus, to obtain a convergent regularization method we further need to choose this parameter – this is typically and most easily done by a parameter choice rule in dependence on the noise level $\delta > 0$.

Indeed, if the regularization parameter $\alpha = \alpha(\delta)$ is chosen suitably in dependence on the noise level δ, then several important results on the convergence of minimizers to $\mathcal{J}_{\alpha,s_m^\delta}$ have been shown, see [SV89, EKN89]. In detail, if α is chosen as a scalar function of δ that satisfies

$$\alpha(\delta) \to 0 \quad \text{and} \quad \frac{\delta^2}{\alpha(\delta)} \to 0 \quad \text{as} \delta \to 0, \tag{5.31}$$

then it is guaranteed that the family of minimizers $p_{\alpha(\delta)}^\delta$ of $\mathcal{J}_{\alpha(\delta),s_m^\delta}$ possesses at least one convergent subsequence as $\delta \to 0$. Moreover, the limit $p^\dagger \in \mathcal{P}$ of any convergent subsequence of these minimizers solves $\Phi(p^\dagger) = s_m$ and furthermore minimizes the functional $\| \cdot - p^* \|_X$ among all solutions to that equation, i.e., $\| p^\dagger - p^* \|_X \leq \| p - p^* \|_X$ for all solutions $p \in \mathcal{P}$ to $\Phi(p) = s_m$; $p^\dagger$ is henceforth called a minimal-norm-solution. If there exists only one minimal-norm-solution $p^\dagger$, then $p_{\alpha(\delta)}^\delta$ converges to $p^\dagger$ as $\delta > 0$ tends to 0 (no matter which of the possibly several minimizers for fixed $\delta > 0$ is chosen); if $p^\dagger$ further satisfies source conditions, see [EKN89, EHN96], then this convergence further satisfies an algebraic convergence rate less than one.

We emphasise that it is unrealistic to expect more than convergence of the regularisers $p_{\alpha(\delta)}^\delta$ to a solution of the non-linear equation as the noise level $\delta > 0$ tends to zero – for fixed $\delta > 0$, it is in general impossible to deduce exact information from noisy data. Even if the parameter identification problem was stable, the latter convergence result is the best one can hope for.

Rules for the Choice of the Regularization Parameter

Two problems of the above formulation of Tikhonov regularization and the related convergence results naturally arise: As the convergence result holds asymptotically as $\delta \to 0$, it is unclear how to choose $\alpha = \alpha(\delta)$ in practice for some known noise level (or, even worse, if the noise level δ is unknown). Second, it is not clear how to tackle the minimisation problem (5.30) numerically. While the second point is treated in the subsequent Section 4, we discuss in the following a number of techniques that cope with the problem of suitably choosing the regularization parameter from the first point.

A typical parameter choice method to determine $\alpha = \alpha(\delta)$ if the noise level $\delta > 0$ is known (or can at least be estimated or guessed) is the discrepancy principle of Morozov, see [Mor68]: Fix some monotonically decreasing sequence $\alpha_n \to 0$, compute for each $n = 1,2, \ldots$ a minimiser $p_{\alpha_n}^\delta$ of (5.30), and pick the first minimizer $p_{\alpha_{n^*}}^\delta$ with discrepancy $\| W(\Phi(p_{\alpha_{n^*}}^\delta) - s_m^\delta) \|_{\mathbb{R}^N}$ less than $\tau\varepsilon$ for some fixed parameter $\tau > 1$, i.e.,

ISBN 978-3-527-33606-7

$$\| W\left(\Phi\left(p^{\delta}_{\alpha_{n^*}}\right) - s^{\delta}_{\mathrm{m}}\right) \|_{\mathbb{R}^N} \leq \tau\delta \quad \text{whereas } \| W\left(\Phi\left(p^{\delta}_{\alpha_n}\right) - s^{\delta}_{\mathrm{m}}\right) \|_{\mathbb{R}^N} > \tau\delta \text{ for } n = 1,2, \ldots, n^* - 1. \tag{5.32}$$

Typically, the sequence α_n is chosen as power of $1/n$ or as t^n for some t in the interval $(0,1)$, according to the degree of instability of the identification problem, and τ is chosen in between 1.1 and 4. However, let us mention that in a specific application optimal values of τ in terms of the distance of $p^{\delta}_{\alpha_{n^*}}$ to the true parameter might sometimes be smaller than one.

The parameter choice rule (5.32) is the so-called discrepancy principle, which basically attempts to solve the equation up to the noise level, see [Mor68]. The above-sketched convergence results for the minimizers $p^{\delta}_{\alpha(\delta)}$ hold as well for $p^{\delta}_{\alpha_{n^*}}$ as $\delta \to 0$. Note that a modified version of the above-described principle attempts for find $\alpha(\delta) > 0$ such that the discrepancy satisfies $\| W(\Phi(p^{\delta}_{\alpha(\delta)}) - s^{\delta}_{\mathrm{m}}) \|_{\mathbb{R}^N} = \delta$, see [Kir96]; this variant is particularly attractive for linear problems, as we sketch later on.

If the noise level δ is unknown, several heuristic parameter choices have been proven to work well. We refer first to the L-curve criterion [Han92] which picks α by maximizing the curvature of the so-called L-curve, a curve in two dimensions defined by

$$\alpha \mapsto (\log \| W(\Phi(p^{\delta}_{\alpha}) - s^{\delta}_{\mathrm{m}}) \|_{\mathbb{R}^N}, \| D(p - p^*) \|^2_Z) \in \mathbb{R}^2. \tag{5.33}$$

Second, the quasi-optimality criterion, see [Mor84], picks α as the minimizer of

$$\alpha \mapsto \alpha \| D(p^{\delta}_{\alpha} - p^*) \|_Z. \tag{5.34}$$

Third, generalized cross-validation, see [Wah90], is a heuristic parameter choice rule that applies for linear Φ and $q = 2$. For simplicity, we present this rule in the case $X = \mathbb{R}^L$ and assume first that F is a matrix representation of Φ with transpose matrix F^T, i.e., $\Phi(p) = Fp$ and $Fp \cdot s = p \cdot F^T s$ for all $p \in X = \mathbb{R}^L$ and all $s \in S = \mathbb{R}^N$, and that both W and D are the identity and can hence be omitted in (5.30). Under these assumptions, generalized cross-validation picks α as minimizer of

$$\alpha \mapsto \frac{\|Fp^{\delta}_{\alpha} - s^{\delta}_{\mathrm{m}}\|^2_{\mathbb{R}^N}}{[trace(I - F(F^TF + \alpha^2 I)^{-1}F^T)]^2}. \tag{5.35}$$

(Here, I denotes the identity matrix and $trace$ is the sum of the diagonal elements of a matrix.) Both the L-curve and the quasi-optimality criterion can be interpreted as a tool to balance the discrepancy against the norm of the reconstructed parameter; the generalized cross-validation technique stems from statistics and can be interpreted as yielding a risk-minimizing ridge estimator for a solution to $\Phi(p) = s^{\delta}_{\mathrm{m}}$, see [GHW79].

Explicit Minimizers for Linear Models

If Φ is a linear mapping, if $q = 2$, and if the set of admissible parameters $\mathcal{P}$ equals the entire linear space X, then the minimizer to (5.30) is unique and can be computed explicitly by solving a linear matrix-vector equation. To this end, let us assume that the minimization problem under consideration is already discretized, such that $X = \mathbb{R}^L$ and that $F: \mathbb{R}^L \to \mathbb{R}^N$ is a matrix representation of Φ, i.e., $\Phi(p) = Fp$ for all $p \in X = \mathbb{R}^L$. The Tikhonov minimization problem (5.30) hence is to find a solution $p_\alpha^\delta \in \mathbb{R}^L$ to the minimization problem

$$p_\alpha^\delta = \min_{p \in \mathbb{R}^L} \mathcal{J}_{\alpha, s_m^\delta}(p), \qquad \mathcal{J}_{\alpha, s_m^\delta}(p) = \| W(Fp - s_m^\delta) \|_{\mathbb{R}^N}^2 + \alpha \| D(p - p^*) \|_{\mathbb{R}^M}^2, \tag{5.36}$$

where, by abuse of notation, we assume that D is an injective matrix in $\mathbb{R}^{M \times L}$ with $M \geq L$. Computing the Lagrange equation of the functional on the right of (5.30) shows that the minimiser p_α^δ solves the linear system of equations

$$[F^T W^T W F + \alpha D^T D] p_\alpha^\delta = F^T W^T s_m^\delta + \alpha D^T D p^*, \tag{5.37}$$

see, e.g., [Kir96, EHN96]. The latter matrix-vector equation is always uniquely solvable because the matrix $F^T W^T W F + \alpha D^T D$ is positive definite and hence invertible on $\mathbb{R}^L$ for all $\alpha > 0$.

Remark 3

*If X is any finite-dimensional linear vector space, not necessarily equal to $\mathbb{R}^L$, and if Φ is a linear mapping, then a similar linear equation as (*5.37*) can also be set up for the minimizer of p_α^δ of $\mathcal{J}_{\alpha, s_m^\delta}$. As X possesses a finite basis $\{b_1, \ldots, b_L\}$, every $p \in X$ can be represented as*

$$p = \sum_{j=1}^L v_j b_j \quad \text{for a unique vector } \mathrm{v} = \mathrm{v_p} = (\mathrm{v_j})_{\mathrm{j=1}}^{\mathrm{L}} \in \mathbb{R}^{\mathrm{L}}. \tag{5.38}$$

(When working with, e.g., a finite-element discretization, the b_j are the finite element basis functions defined on a mesh.) The matrix F is then defined by $Fv = \Phi(\sum_{j=1}^L v_j b_j)$ for all $v \in \mathbb{R}^L$. However, expressing the norm of $D(p_\alpha^\delta - p^*)$ in Z via the norm of the coefficient vector of $p - p^*$ requires to introduce a suitable Gram matrix, which we omit here, referring to Chapter 9 in [EHN96] or Chapter 6 in [Rie03].

Remark 4

(1) Many results on linear models Φ in this section can be reformulated using the singular value decomposition of the matrix $WF \in \mathbb{R}^{N \times L}$, see, e.g., [EHN96, Kir96]; we skip details to avoid the need to introduce this matrix decomposition.

(2) Several iterative solution methods for linear systems as (5.37) provide fast alternatives to solving these systems by a direct solver as, e.g., Gaussian elimination and its variants.

ISBN 978-3-527-33606-7

Such algorithms include the conjugate gradients method, the GMRES (generalized minimal residual) method and the steepest descent method, see [Kel95].

In the same latter linear setting, there is an efficient way to compute $\alpha = \alpha(\delta)$ such that the modified discrepancy principle $\| W(F(p^{\delta}_{\alpha(\delta)}) - s^{\delta}_{m}) \|_{\mathbb{R}^N} = \delta$ introduced above is satisfied: Since the scalar function

$$\gamma(\alpha) = \| W(F(p^{\delta}_{\alpha}) - s^{\delta}_{m}) \|^2_{\mathbb{R}^N} - \delta^2 \tag{5.39}$$

is continuously differentiable and monotonously decreasing, Newton's method allows to find a positive zero of γ. The derivative of γ equals

$$\gamma'(\alpha) = 2(F^T W^T (WF(p^{\delta}_{\alpha}) - s^{\delta}_{m})) \cdot \frac{\mathrm{d}p^{\delta}_{\alpha}}{\mathrm{d}\alpha} = -2\alpha(D^T D(p^{\delta}_{\alpha} - p_0)) \cdot \frac{\mathrm{d}p^{\delta}_{\alpha}}{\mathrm{d}\alpha}, \tag{5.40}$$

where we denoted the derivative of the vector p^{δ}_{α} with respect to α by $\mathrm{d}p^{\delta}_{\alpha}/\mathrm{d}\alpha$. This derivative is again a vector in $\mathbb{R}^L$ and satisfies a linear equation with the same matrix on the left-hand side as in (**5.37**),

$$[F^T W^T WF + \alpha D^T D] \frac{\mathrm{d}p^{\delta}_{\alpha}}{\mathrm{d}\alpha} = -D^T D(p^{\delta}_{\alpha} - p_0),$$

such that computing this derivative allows to evaluate $\gamma'(\alpha)$ via (5.40). The Newton iteration to determine $\alpha = \alpha(\delta) > 0$ such that $\| W(F(p^{\delta}_{\alpha(\delta)}) - s^{\delta}_{m}) \|_{\mathbb{R}^N} = \delta$ hence reads

$$\alpha_{n+1} = \alpha_n - \frac{\gamma(\alpha_n)}{\gamma'(\alpha_n)}, \qquad n = 1,2,3 \ldots.$$

If $p_0 = 0$ and $\| s^{\delta}_{m} \|_{\mathbb{R}^N} > \delta$, then a suitable initiation is $\alpha_0 = \delta \| WF \|^2_{\mathbb{R}^L \to \mathbb{R}^N} / (\| s^{\delta}_{m} \|_{\mathbb{R}^N} - \delta)$; if $\| s^{\delta}_{m} \|_{\mathbb{R}^N} \leq \delta$, then obviously $p = 0$ already satisfies the discrepancy principle.

The Soft-Shrinkage Iteration

If one considers, in the same linear setting, the following Tikhonov functional

$$\mathcal{J}_{\alpha, s^{\delta}_{m}}(p) = \| W(Fp - s^{\delta}_{m}) \|^2_{\mathbb{R}^N} + \alpha \sum_{j=1}^{L} |p_j - p_{0,j}|^q \qquad \text{for } p \in \mathbb{R}^L \text{ and fixed } q \geq 1, \tag{5.41}$$

with different penalty term, then explicit formulas for minimizers $p^{\delta}_{\alpha} \in \mathbb{R}^L$ to this functional do not exist. Still, one can write down a fixed-point equation for p^{δ}_{α} that is particularly popular in the case $q = 1$, where it is known as the soft-shrinkage iteration, see [DDD04], particularly attractive if the searched-for parameter differs from p_0 merely

in few entries. The announced fixed-point iteration relies on the scalar function $f_{\alpha,q}\colon \mathbb{R} \to \mathbb{R}$, defined by

$$f_{\alpha,q}(t) = t + \alpha q sign(t)|t|^{q-1}, \qquad \text{for } t \in \mathbb{R}, q \geq 1, \alpha > 0.$$

If $q > 1$, the function $t \mapsto sign(t)\,|t|^{q-1}$ is monotonically increasing and bijective from $\mathbb{R}$ into $\mathbb{R}$, such that the function $f_{\alpha,q}$ is invertible. For simplicity, we denote its inverse function by $s_{\alpha,q}$. This inverse function determines the q-shrinkage operator $S_{\alpha,q}\colon \mathbb{R}^L \to \mathbb{R}^L$, defined for $p \in \mathbb{R}^L$ element-wise by

$$S_{\alpha,q}(p) = w \in \mathbb{R}^L, \qquad w_j = \sum_{j=1}^{L} s_{\alpha,q}(p_j) \quad \text{for} j = 1, \dots, N.$$

The shrinkage operator is also well-defined for $q = 1$, where one can even express its action on p explicitly, using the function

$$s_{\alpha,1}(t) = \begin{cases} t + \alpha/2 & \text{if } t \leq -\alpha/2, \\ 0 & \text{if } |t| < \alpha/2, \\ t - \alpha/2 & \text{if } t \geq \alpha/2. \end{cases}$$

For $q \geq 1$, one can then show that any minimizer p_α^δ to the functional in (5.41) satisfies

$$p_\alpha^\delta - p_0 = S_{\alpha,q}\left(p_\alpha^\delta - p_0 + F^T W^T (s_{\mathrm{m}}^\delta - W F p_\alpha^\delta)\right).$$

Consequently, the corresponding fixed-point iteration reads

$$p_{n+1} = p_0 + S_{\alpha,q}\left(p_n - p_0 + F^T W^T (s_{\mathrm{m}}^\delta - W F p_n)\right), \quad n \in \mathbb{N}, \tag{5.42}$$

for an arbitrary starting vector $p_0 \in \mathbb{R}^L$, and converges to a minimizer p_α^δ of $\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}$ defined in (5.41). If one couples the regularization parameter $\alpha = \alpha(\delta)$ with the noise level δ as in (5.31), the minimizers again converge to the minimum-norm solution to $Fp = s_{\mathrm{m}}$ (respectively, to $\Phi(p) = s_{\mathrm{m}}$) as $\delta \to 0$.

Of course, one can also stop this iteration by the discrepancy principle whenever the nth iterate p_n from (5.42) satisfies

$$\| Fp_n - s_{\mathrm{m}}^\delta \| \leq \tau\varepsilon \qquad \text{for some fixed } \tau > 1. \tag{5.43}$$

The discrepancy principle is particularly suited for iterative regularization schemes such as (5.42), as little extra work apart from the iteration itself is necessary to set up the corresponding regularization scheme. Alternatively, heuristic parameter choice rules can

ISBN 978-3-527-33606-7

also be employed; we refer to [HPR09, HPR11] for the quasi-optimality criterion applied to iterative schemes and a comparison of several parameter choice rules for both iterative and non-iterative regularization methods.

5.3.4 Iterative Regularization Schemes

If the model Φ is non-linear, then there are in general no explicit representations for the minimizer of the Tikhonov functional given in (**5.30**). If one aims to rely on Tikhonov regularization in a non-linear setting, one hence needs to compute this minimizer by some minimization scheme suitable for functionals as given in (**5.29**). Typically, such schemes are of iterative nature and frequently they rely on the differentiability of Φ with respect to the parameter p: By definition, $\Phi: \mathcal{P} \subset X \to \mathcal{S} = \mathbb{R}^N$ is differentiable at $p \in \mathcal{P}$ if there is a linear mapping $\Phi'(p): X \to \mathbb{R}^N$ such that

$$\frac{\|\Phi(p+h)-\Phi(p)-\Phi'(p)h\|_{\mathbb{R}^N}}{\|h\|_X} \to 0 \quad \text{as } \|h\|_X \to 0.$$

In a setting where X (and possibly also $\mathcal{S}$) are infinite-dimensional spaces, this concept is called Fréchet differentiability; in finite-dimensional spaces, this is nothing but the well-known differentiability of functions that map $\mathbb{R}^L$ into $\mathbb{R}^N$.

Gradient Descent Schemes

Let us assume from now on that Φ is differentiable at all parameters $p \in \mathcal{P}$. Then, the derivative $\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}'(p)$ of the Tikhonov functional

$$\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}(p) = \| W(\Phi(p) - s_{\mathrm{m}}^\delta) \|_{\mathbb{R}^N}^2 + \alpha \| D(p - p^*) \|_Z^2$$

from (10) with $q = 2$ at p, can be explicitly computed by the chain rule,

$$\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}'(p)h = 2\big(\Phi'(p)h, W^T W(\Phi(p) - s_{\mathrm{m}}^\delta)\big)_X + 2(h, \alpha D^T D(p - p^*))_X. \tag{5.44}$$

Here, $D^T: Z \to X$ is the transpose mapping to D (i.e., the transpose matrix, if $D \in \mathbb{R}^{M \times L}$ is a matrix), defined by $(p, D^T z)_X = (Dp, z)_Z$ for all $p \in X$ and $z \in Z$. If we analogously denote the transpose mapping of $\Phi'(p)$ by $\Phi'(p)^T: \mathbb{R}^N \to X$, then (**5.44**) implies that $-\Phi'(p)^T W^T W(\Phi(p) - s_{\mathrm{m}}^\delta) - \alpha D^T D(p - p^*)$ is a descent direction of the functional $\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}$ at p. This motivates the following iteration for the minimization of $\mathcal{J}_{\alpha,s_{\mathrm{m}}^\delta}$,

$$\begin{aligned} p_{n+1}^\delta &= p_n^\delta - \omega_n \big[\Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) + \alpha D^T D(p_n^\delta - p^*)\big] \\ &= (I - \omega_n \alpha D^T D) p_n^\delta - \omega_n \big[\Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) - \alpha D^T D p^*\big], \quad n \in \mathbb{N}, \end{aligned} \tag{5.45}$$

for some starting value (or initial guess) $p_0 \in \mathcal{P}$. (Here, $I: p \mapsto p$ is the identity mapping on X, that is, the identity matrix in a discretized setting.) The numbers $\omega_n > 0$ can be interpreted as a step size control or a damping parameter that should be chosen such that both linear mappings $\omega_n \Phi'(p_n^\delta)^T W^T W$ and $\omega_n \Phi'(p_n^\delta)^T W^T W$ are contractive, that is,

$$\omega_n \| \Phi'(p_n^\delta)^T W^T W s \|_X < \| s \|_{\mathbb{R}^N} \quad \text{and} \quad \omega_n \alpha \| D^T D p \|_X < \| p \|_X \quad \text{for all} s \in \mathbb{R}^N, p \in \mathcal{P}.$$

In other words, the spectral matrix norm (i.e., the 2-norm) of the matrix representations of both mappings needs to be strictly less than one for each $n \in \mathbb{N}$, see [HNS95, KNS08]. Typically, the latter condition is far from being easy to guarantee in applications, in particular if it is costly to set up the entire matrix representation of $\Phi'(p_n^\delta)^T$.

A related method is the so-called non-linear Landweber iteration, where one chooses the update direction as a descent direction of the squared discrepancy $\| W(\Phi(p) - s_{\mathrm{m}}^\delta) \|_{\mathbb{R}^N}^2$ (neglecting the penalty term of $\mathcal{J}_{\alpha, s_{\mathrm{m}}^\delta}$), such that the resulting iteration reads

$$p_{n+1}^\delta = p_n^\delta - \omega_n \Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta), \quad n \in \mathbb{N}. \tag{5.46}$$

Again, this iteration scheme starts with some initial guess $p_0 \in \mathcal{P}$. The damping parameter $\omega_n > 0$ has to be chosen so small that $\omega_n \Phi'(p_n^\delta)^T W^T W$ is contractive, that is,

$$\omega_n \| \Phi'(p_n^\delta)^T W^T W s \|_X < \| s \|_{\mathbb{R}^N} \qquad \text{for all } s \in \mathbb{R}^N, n \in \mathbb{N}.$$

Both iterations () and () are connected, which becomes evident if one replaces the term $D^T D(p_n^\delta - p^*)$ in (25) by $D^T D(p_{n+1}^\delta - p^*)$ and chooses $p^* = 0$. The resulting iteration then reads

$$(I - \omega_n \alpha D^T D) p_{n+1}^\delta = p_n^\delta - \omega_n \Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta), \qquad n \in \mathbb{N},$$

which is a Landweber iteration subject to an additional smoothing by multiplying the Landweber iterate with the inverse of $I - \alpha D^T D$,

$$p_{n+1}^\delta = (I - \omega_n \alpha D^T D)^{-1} \left[p_n^\delta - \omega_n \Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) \right], \qquad n \in \mathbb{N}.$$

(The inverse $(I - \omega_n \alpha D^T D)^{-1}$ exists if ω_n is for all $n \in \mathbb{N}$ so small that the matrix representation of $\omega_n \alpha D^T D$ has a (spectral) matrix norm less than one.) Typically, the numbers ω_n are either all fixed to $\omega > 0$ small enough (this requires to know the magnitude of Φ' in advance), or chosen adaptively by some step size rule that aims to minimize the value of the Tikhonov functional (for (**5.46**)) or of the discrepancy (for (**5.46**)) along

ISBN 978-3-527-33606-7

the descent direction. Examples for step size rules include the bisection and the Armijo rule, see [Arm66, BB88].

As (**5.46**) attempts to minimize the Tikhonov functional, whereas (**5.47**) attempts to minimize the discrepancy, a crucial difference between the iterations (**5.46**) and (**5.47**) is that the fixed point equations satisfied by any limit $p^\dagger$ of the sequence of iterates $\{p_n^\delta\}$ differ,

$$0 = \Phi'(p^\dagger)^T W^T W(\Phi(p^\dagger) - s_{\mathrm{m}}^\delta) + \alpha D^T D(p^\dagger - p^*) \quad \text{for (Error! Re}$$

$$0 = \Phi'(p^\dagger)^T W^T W(\Phi(p^\dagger) - s_{\mathrm{m}}^\delta) \quad \text{for (Error! Reference source not found.)(Erro}$$

If we assume that $\Phi'(p^\dagger)^T$ is injective, then any limit of the Landweber iteration (**5.47**) hence satisfies the equation $\Phi(p^\dagger) = s_{\mathrm{m}}^\delta$; obviously, this does not hold for the limit to (**5.45**). Thus, computing a minimizer p_α^δ of $\mathcal{J}_{\alpha, s_{\mathrm{m}}^\delta}$ via the iteration (**5.45**) must be accompanied by a parameter choice rule to determine a value of α that ensures both stability and accuracy of the resulting parameter reconstruction, see (**5.32**)-(**5.35**). (Several of these rules require to compute several minimizers, such that computation time might become a crucial point here; whenever run-time becomes an issue, experience on suitable choices of α is typically required to reduce computation times.)

By construction, the Landweber iteration (**5.46**) attempts to compute a solution to the equation $\Phi(p) = s_{\mathrm{m}}^\delta$ instead of a minimizer of $\mathcal{J}_{\alpha, s_{\mathrm{m}}^\delta}$. We discussed in Section 1 that solving ill-conditioned operator equations exactly leads to incorrect results due to instability; thus, the Landweber iteration must in any case be stopped by some stopping rule that ensures stability and accuracy of the resulting approximation to the parameter p. (Such a stopping rule hence plays the same role as the regularization parameter choice rule for Tikhonov regularization.) If the noise level δ is known, a particularly attractive stopping rule is the discrepancy principle that we already discussed for Tikhonov regularization in (**5.32**). For an iterative regularization method, Morozov's discrepancy principle stops the iteration whenever the discrepancy is below the tolerance $\tau\delta$ for the first time: The resulting approximation to the true parameter is hence $p_{n^*}^\delta$, determined by

$$\| W(\Phi(p_{n^*}^\delta) - s_{\mathrm{m}}^\delta) \|_{\mathbb{R}^N} \leq \tau\delta \quad \text{whereas} \quad \| W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) \|_{\mathbb{R}^N} > \tau\delta \text{for} n = 1, 2, \ldots, n^* - 1. \tag{5.47}$$

Again, $\tau > 1$ is a fixed parameter, typically chosen between 1.1 and 4. (Suitable values for a specific application should be tested and optimized.) Other stopping rules for known noise level suitable for the Landweber iteration include the quasi-optimality criterion and Lepskii's balancing principle; if we assume that $X = \mathbb{R}^L$ such that $\Phi'(p_n^\delta)^T$ is a matrix in $\mathbb{R}^{L \times N}$, the these two rules stop the iteration at the nth iterate p_n^δ if

$$[\Phi(p_n^\delta) - \Phi(p_{n+1}^\delta)] \cdot \Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) \leq 2\delta \parallel \Phi'(p_n^\delta)^T W^T W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) \parallel_X \tag{5.48}$$

and if

$$\parallel p_n^\delta - p_{n+1}^\delta \parallel \geq \tau \max\{\omega_1^{1/2}, \ldots, \omega_n^{1/2}\} n^{1/2} \delta, \tag{5.49}$$

respectively. These noise level-free rules hence stop the Landweber iteration whenever the discrepancy is small enough or if the difference between two iterates becomes too large compared to the iteration number. Notably, both rules require to compute $n^* + 1$ iterates for the determination of the stopping index n^*, that is, one iterate more than required by the discrepancy principle. A further heuristic parameter choice rule applicable to the Landweber iteration is the Hanke-Raus rule. This stopping rule determines the stopping index rule n^* as a minimizer of the sequence of numbers $\sqrt{n} \parallel W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta) \parallel_{\mathbb{R}^N}$ in n; the resulting parameter approximation is hence $p_{n^*}^\delta$, see [HR96, HPR09, HPR11].

Both iterative schemes (**5.46**) and (**5.47**) can also be used to regularise *linear* problems, where Φ can be represented by some matrix F. In this case, one usually fixes the damping parameter α_n to $\alpha > 0$; the upper bound $\alpha < 2/\parallel WF \parallel^2$ involves the spectral matrix norm $\parallel WF \parallel^2$ of WF is crucial for convergence of the iteration.

There is a numerous works concerned with the convergence and regularization properties of the Landweber iteration both for linear and non-linear models Φ, see [Lan51, HNS95]. In particular, the following convergence result holds for non-linear Φ if the iteration is stopped by the discrepancy principle, returning $p_{n^*}^\delta$ as reconstruction: If $\Phi'(p)$ is injective for all $p \in \mathcal{P}$ and if the initial value of the iteration is close enough to any solution to $\Phi(p) = s_{\mathrm{m}}^\delta$, then $p_{n^*}^\delta$ converges to a solution $p^\dagger$ to that equation as $\delta \to 0$. For linear mappings Φ, this convergence result holds independently of the choice of the initial value of the Landweber iteration, at least if Φ is injective, see [EHN96] for details. Similar as for Tikhonov regularization, convergence rates in terms of δ can be shown if the starting value p_0 solution satisfies a source condition, see [Han95]. (For non-linear Φ, convergence of Landweber-type regularization methods typically rely on non-linearity conditions for Φ; if $\Phi'(p)$ is injective for all $p \in \mathcal{P}$, these conditions are automatically satisfied in our setting as X is finite dimensional.).

Newton-type Regularization Schemes

The Landweber iteration has the advantage to be rather robust against data noise; on the downside, this method is rather slow and cannot be advertised for small noise levels (for linear models, the number of iteration steps for constant damping parameter behaves like δ^{-2}). For this reason, a different class of iterative Newton-like methods is usually preferred, in particular for rather accurate data. Instead of trying to find stable solutions to an

ISBN 978-3-527-33606-7

equation for a searched-for parameter by minimizing a functional, these methods try to approximately solve the equation $\Phi(p) = s_m^\delta$ by successive linearizion: At each iterate p_n^δ one considers the linearizion of Φ at p_n^δ,

$$\Phi(p) \approx \Phi(p_n^\delta) + \Phi'(p_n^\delta)h \qquad \text{if } \| \, h \, \|_X \text{ is sufficiently small.}$$

Aiming to determine an update $h_n^\delta \in X$ that defines the next iterate $p_{n+1}^\delta = p_n^\delta + h_n^\delta$ such that $\Phi(p_{n+1}^\delta) = s_m^\delta$ hence leads to the following linear equation for $h_n^\delta \in X$,

$$\Phi'(p_n^\delta)h_n^\delta = s_m^\delta - \Phi(p_n^\delta). \tag{5.50}$$

For parameter identification problems, this linearized equation typically cannot stably inverted either, such that a linear regularization scheme must be applied to (5.30). Let us opt here for Tikhonov regularization and denote, as above in (**5.45**), by $\Phi'(p_n^\delta)^T : \mathbb{R}^N \to X$ the transpose mapping of the linear map $\Phi'(p_n^\delta)$. Further, let us first choose the a-priori information p^* from the Tikhonov functional in (5.30) or (5.36) to be zero. Due to the results from the last section, see (16), the minimizer $h_n^\delta \in X$ of the Tikhonov functional

$$\mathcal{J}_{\alpha_n, s_m^\delta - \Phi(p_n^\delta)}(h) = \| \, W\left(\Phi'(p_n^\delta)h - s_m^\delta + \Phi(p_n^\delta)\right) \|_X^2 + \alpha_n \, \| \, Dh \, \|_Z^2 \quad \text{on } X$$

for some regularization parameter $\alpha_n > 0$, a matrix $W \in \mathbb{R}^{N \times N}$ and an injective map D from X into the finite-dimensional space Z satisfies the linear equation

$$\left[(\Phi'(p_n^\delta))^T W^T W \Phi'(p_n^\delta) + \alpha_n D^T D\right] h_n^\delta = (\Phi'(p_n^\delta))^T W^T (s_m^\delta - \Phi(p_n^\delta)), \quad n \in \mathbb{N}, \tag{5.51}$$

which allows for a straightforward computation of h_n^δ if $X = \mathbb{R}^L$, since all involved quantities then become matrices or vectors. (We omitted the dependence of h_n^δ on α_n.) In particular, h_n^δ is the *unique* minimiser of that functional. The resulting iterative regularisation technique with iterates $\{p_j^\delta\}$ is the Levenberg-Marquardt method, see [Lev44, Mar63].

Again, we need to discuss the choice of the regularization parameters α_n, as well as the choice of a stopping rule for the Levenberg-Marquardt iteration. Concerning the first issue, it is simplest to choose α_n as a sequence of positive numbers that tends strictly monotonically to zero, such as $\alpha_n = n^{-r}$ for some $r > 0$ or $\alpha_n = r^n$ for some r in the interval $(0,1)$. Computationally more involved are inexact Newton methods that determine $\alpha_n > 0$ such that the solution $w_n^\delta = w_{n,\alpha_n}^\delta$ to (5.51) satisfies the linearized equation up to a tolerance $\mu \in (0,1)$, see [Han97],

$$\| W\left(\Phi'(p_n^\delta)h_{n,\alpha_n}^\delta - s_{\mathrm{m}}^\delta + \Phi(p_n^\delta)\right)\|_{\mathbb{R}^N} \leq \mu \| \Phi(p_n^\delta) - s_{\mathrm{m}}^\delta \|_{\mathbb{R}^N} \quad \text{for some fixed } \mu \in (0,1). \tag{5.52}$$

One typically chooses μ in between 0.01 and 0.3; the techniques we used in (5.32) or (5.39) to find a suitable regularization parameter by the discrepancy principle can be used here to find a suitable $\alpha_n > 0$ such that w_{n,α_n}^δ satisfies (5.52). Of course, this implies extra computational work as w_{n,α_n}^δ possibly needs to be computed for several choices of α_n until (5.52) is satisfied.

To determine the stopping index n^* of the sequence of iterates p_n^δ of the Levenberg-Marquardt method, one can either use the discrepancy principle (5.47), the quasi-optimality criterion (5.48), or Lepskii's balancing principle (5.49) if the noise level $\delta > 0$ is known; otherwise, the heuristic stopping rule indicated above for the Landweber iteration can be exploited: Determine n^* as a the index where the sequence of numbers $\sqrt{n}\,\| W(\Phi(p_n^\delta) - s_{\mathrm{m}}^\delta)\|_{\mathbb{R}^N}$ assumes its minimal value.

Termination and convergence of the Levenberg-Marquardt method combined with the discrepancy principle can for instance be shown if $\Phi'(p)$ is injective for all $p \in \mathcal{P}$, see [Han97] or the monographs [KNS08]. Of course, also other techniques instead of Tikhonov regularization can be used for computing a regularized step h_n^δ; we refer to [LR09] for several other applicable linear schemes as the steepest descent or the conjugate gradients method.

While the Levenberg-Marquardt method chooses the a-priori information p^* from the Tikhonov functional in (5.36) equal to zero, the iteratively regularized Gauss-Newton method, see [Bak92, BNS97], introduces an a-priori guess $p^* \in \mathcal{P}$ and forces the iterates p_n^δ to remain close to p^*. This can be interpreted as incorporation of a-priori information on the searched-for parameter, or as a tool to increase stability of the iteration. The update $h_n^\delta \in X$ in the nth step $p_{n+1}^\delta = p_n^\delta + h_n^\delta$ of this algorithm is, to this end, defined as the unique minimizer of the functional

$$\mathcal{J}_{\alpha_n, s_{\mathrm{m}}^\delta - \Phi(p_n^\delta)}(h) = \| W\left(\Phi'(p_n^\delta)h - s_{\mathrm{m}}^\delta + \Phi(p_n^\delta)\right)\|_{\mathbb{R}^N}^2 + \alpha_n \| D\left(h + p_n^\delta - p^*\right)\|_Z^2 \quad \text{on } X.$$

Equivalently, $h_n^\delta \in X$ satisfies the linear equation

$$\left[(\Phi'(p_n^\delta))^T W^T W \Phi'(p_n^\delta) + \alpha_n D^T D\right] h_n^\delta = (\Phi'(p_n^\delta))^T W^T (s_{\mathrm{m}}^\delta - \Phi(p_n^\delta)) + \alpha_n (p^* - p_n^\delta),$$

that becomes a matrix-vector equation upon discretization, that is, once $X = \mathbb{R}^L$ is a Euclidean space. If α_n is a sequence of positive numbers tending to zero, such that the

ISBN 978-3-527-33606-7

quotient α_n/α_{n+1} is bounded in $n \in \mathbb{N}$ (the above choice $\alpha_n = t^n$ for $0 < t < 1$ satisfies this requirement), then convergence results for the iteratively regularized Gauss-Newton method can be shown for instance if $\Phi'(p)$ is injective for all $p \in \mathcal{P}$..

5.3.5 Numerical Examples in Load Reconstruction

In this last section, we present load reconstructions using two of the above-introduced regularization techniques in the context of load reconstruction, as introduced in Example (1). More precisely, we consider Tikhonov regularization and the Landweber iteration for load monitoring using sensors attached to the lower surface of a thin horizontal plate Ω. The plate Ω of size 0.5m $\times$ 0.5m $\times$ 0.02m is assumed to consist of homogeneous construction steel with elastic modulus E of $210\ \mathrm{kN/mm^2}$ and Poisson's ration ν of 0.3. The Lamé parameters hence equal $\lambda = \nu E/[(1+\nu)(1-2\nu)] = 63/52\mathrm{kN/mm^2}$ and $\mu = E/(2+2\nu) = 1050/13\ \mathrm{kN/mm^2}$. Due to homogeneity of Ω, the rather involved stiffness tensor in (2) hence can be characterized by these two constants.

We additionally assume that one of the four horizontal sides of Ω, denoted by Γ_0, is fixed, and that the top surface Γ_+ is loaded by some space-dependent loading described by a force of the form $-(0,0,f)^T$ for a non-negative scalar function $f\colon \Gamma_+ \to \mathbb{R}$. This loading function is further assumed to be small enough for that the equations of linear elasticity provide a valid model. Under these assumptions, the deformation field $u\colon \Omega \to \mathbb{R}^3$ is governed by the following boundary value problem for the Cauchy stress tensor $\sigma(u) = 2\mu\varepsilon(u) + \lambda div\, u\ I_3$,

$$div\,\sigma(u) = 0 \text{ in } \Omega, \quad u = 0 \text{ on } \Gamma_0, \qquad \sigma(u)\nu = (0,0,-f)^T \text{ on } \Gamma_+, \quad \sigma(u)\nu = 0 \text{ on } \partial\Omega\backslash(\Gamma_+ \cup \Gamma_0). \tag{5.53}$$

(Recall from (1) that $2\varepsilon(u) = \nabla u + (\nabla u)^T$ and that ν is the exterior unit normal field to the boundary $\partial\Omega$.) Due to the Dirichlet boundary condition on Γ_0, there is a unique deformation field that solves the latter boundary value problem. Abbreviating $A\colon B = \sum_{i,j=1}^3 A_{i,j}B_{i,j}$, an integration by parts implies the following associated variational formulation for the solution u,

$$\int_\Omega [2\mu\varepsilon(u)\colon\varepsilon(u) + \lambda div\, u\, div\, v]x = \int_{\Gamma_+} f \cdot v_3 S \tag{5.54}$$

for all sufficiently smooth functions $v\colon \Omega \to \mathbb{R}^3$. This variational formulation allows to approximate u by a finite element method, replacing u and v in (5.54) by, e.g., piecewise polynomial and globally continuous vectorial finite element functions on a triangulation of Ω. For our examples presented below, we simulated loadings by piecewise quadratic elements on a regular tetrahedral mesh with mesh width $1/10$ mm.

Before explaining details on the simulated measurements, let us remark that the proceedings paper [BL14] and the preprint **[BOS16A]** contain several numerical example of loads reconstructed from sensor network data in a similar setting. Extending the examples below, these references further include particular inversion algorithms for linear problems as the conjugate gradients method and reconstructions from highly imprecise data due to incomplete data propagation through a self-organizing sensor network.

We next assume that the sensors attached to the lower surface the plate measure surface strain in x- and y-direction at 64 sensor positions x_j on a grid of sensors of size 8×8. Such measurements can for instance be obtained from strain gauges; we crudely model them by the first two diagonal entries $\partial u/\partial x$ and $\partial u/\partial y$ of the strain tensor $\varepsilon(u)$ at the sensor positions.

Thus, we obtain a physical model Φ linking the applied loading with surface strain measurements to (5.53),

$$\Phi f = \left(\frac{\partial u}{\partial x}(x_j), \frac{\partial u}{\partial y}(x_j)\right)_{j=1}^{64} \subset \mathbb{R}^{128}.$$

For simplicity, we choose f in the space of piecewise constant functions on a regular quadrilateral mesh of 20×20 squares covering Γ_+ completely, see Figure **5.36**(a) for a plot of the mid-points of these squares. The forward mapping Φ thus maps the parameter space $\mathcal{S} = \mathbb{R}^{400}$ into $\mathbb{R}^{128}$. As u and $\varepsilon(u)$ depend linearly on f through (5.53), such that Φ is a linear mapping that can be represented by a matrix F in $\mathbb{R}^{128\times400}$. Using the above-described finite element discretization we simulate the 400 loadings corresponding to the standard basis in $\mathcal{S} = \mathbb{R}^{400}$ to get an explicit approximation to the matrix F representing Φ More precisely, the forces corresponding to these loadings vanish on all but one square of the mentioned surface quadrangulation of Γ_+; on that square, they equal $(0,0,-1)^T$ N/cm^2. As the plate Ω is fixed at one vertical side parallel to the x-axis, strain in x direction is generally smaller than strain in y direction, which can also be observed from the plot of the matrix F in Figure **5.36**(b).

ISBN 978-3-527-33606-7

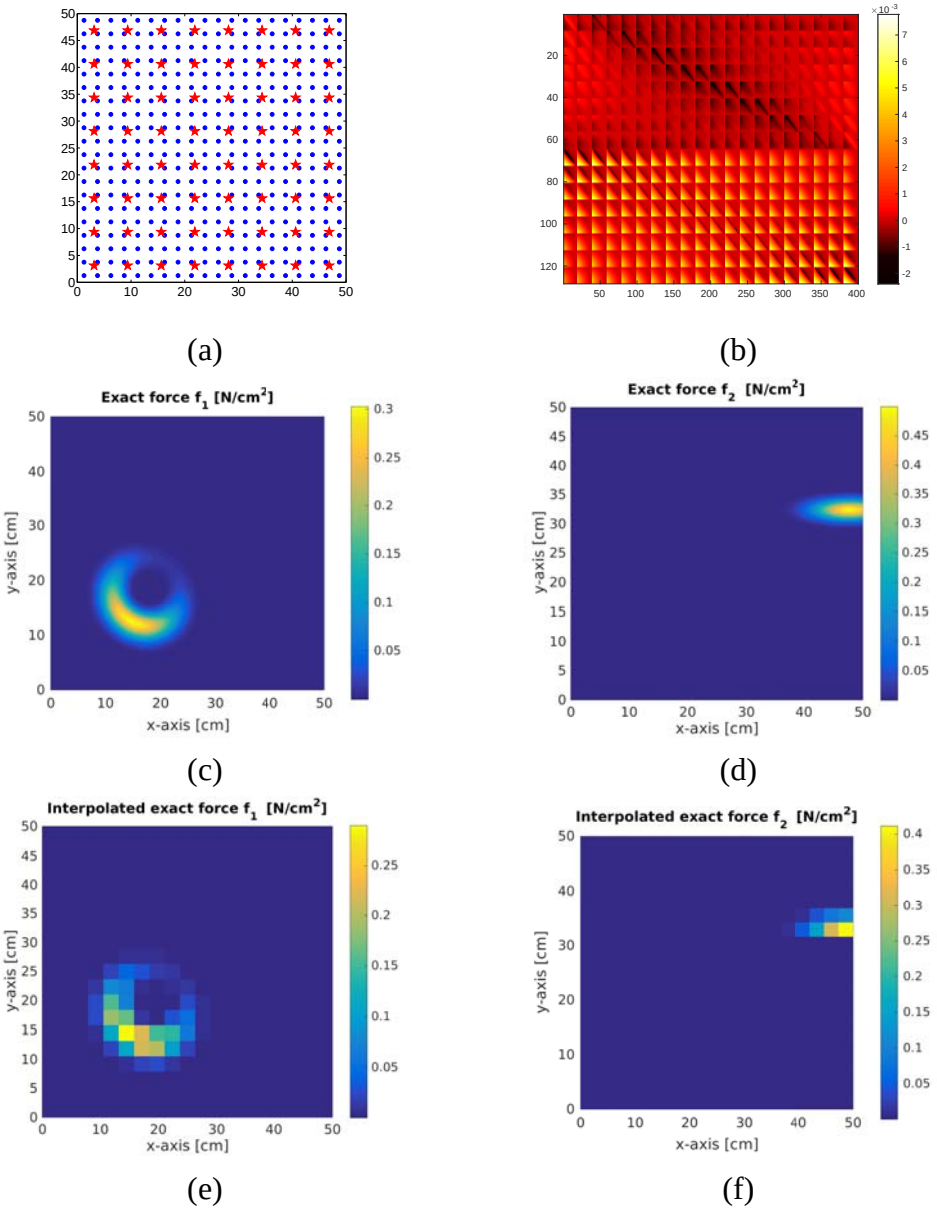


Fig. 5.36 *(a) Positions of the 400 mid points of the squares defining the quadrangulation of Γ_+ for discretized loads f are indicated using blue dots; red crosses indicate the positions of the 64 sensors attached to the lower surface of the plate Ω. As for (c)–(f), units of x- and y-axes are cm. (b) Plot of the load-strain matrix $F \in \mathbb{R}^{128\times400}$. (c)–(d) The loadings $f_{1,2}$. (e)–(f) Piecewise constant interpolations of loadings $f_{1,2}$ in the space of piecewise constant functions on the regular 20×20 grid of the top surface of the plate.*

ISBN 978-3-527-33606-7

To obtain simulated data for inversion by Tikhonov regularization and the Landweber iteration, we further simulate (ideal) surface strain sensor measurements s_{m} for two forces $f_{1,2}$ plotted in Figure **5.36**(c)–(d). Figure **5.36**(e)–(f) shows the piecewise constant interpolations on the 20×20 quadrangulation of Γ^{+}. By checking the convergence of the simulated strain data for different finite-element meshes, we determined that the relative error of the simulated surface strains less than 3%. Nevertheless, we further increase this unknown noise level due to numerical approximation by adding a scaled random matrix containing uniformly distributed numbers in $[-1,1]$ to the simulated data, and choose the scaling such that the resulting relative noise level equals a prescribed relative noise level in between 0.0033 and 0.3. For 100 independent instances of such noised data, we compute reconstructions and present the mean reconstruction and the variance of those 100 reconstructions, to give an impression on the behaviour of both regularization techniques.

For all examples computed by Tikhonov regularization, we choose both mappings W and D as the identity, such that we compute the corresponding regularizes by $p_{\alpha}^{\delta} = (F^{T}F + \alpha I)^{-1}F^{T}s_{\mathrm{m}}^{\delta}$; the regularization parameter is computed by the discrepancy principle (5.32) for $\alpha_n = 10^{2-hn}$ with $h = 0.75$ and $\tau = 1.1$. Note that we work from now on with relative data errors, which are somewhat easier to interpret as misleading scaling effects cannot as easily occur as for absolute errors. The relative noise level of artificially noised data s_{m}^{δ} hence equals $\| s_{\mathrm{m}}^{\delta} - s_{\mathrm{m}} \|_{\mathbb{R}^{128}} / \| s_{\mathrm{m}} \|_{\mathbb{R}^{128}}$, and the relative error of the corresponding reconstruction $f^{\delta} \in \mathbb{R}^{400}$ to the true (interpolated) load $f \in \mathbb{R}^{400}$ equals $\| f^{\delta} - f \|_{\mathbb{R}^{400}} / \| f \|_{\mathbb{R}^{400}}$. Consistently, we also employ the relative discrepancy when checking whether the nth Landweber iterate f_n^{δ} satisfies the discrepancy principle, i.e., whether $\| F(f_n^{\delta}) - s_{\mathrm{m}}^{\delta} \|_{\mathbb{R}^{128}} / \| s_{\mathrm{m}}^{\delta} \|_{\mathbb{R}^{128}} < \tau\delta$.

As we aim to exploit the a-priori knowledge that the exact forces correspond to a loading, and hence point in direction $(0,0,-1)^{T}$, we replace any negative value of any reconstruction by Tikhonov regularisation by zero, and proceed in the same way with any iterate of the Landweber iteration.

Figures **5.37**–**5.40** show the results of Tikhonov regularization and Landweber iteration with regularization parameter and stopping index chosen by the discrepancy principle, respectively, see (5.32) and (5.48). Each figure contains plots for the three different relative noise levels 0.0033, 0.033, and 0.3, ordered row-wise in increasing order. The left column of each figure contains plots of the mean reconstruction f_{mean} computed from all reconstructions due to 100 independent instances $f^{(k)}$ of simulated noisy data, defined by $f_{\mathrm{mean}} = \sum_{k=1}^{100} f^{(k)}/100$. The right column contains plots of the variance of the 100 reconstructions, defined element-wise by $\mathrm{var}(f^{(k)})_j = \sum_{k=1}^{100} |f_j^{(k)} - f_{\mathrm{mean},j}|^2/100$. Notably, the variances of both regularization schemes are so small that it is actually unnecessary to plot to best or the worst reconstruction, as those agree in the visual norm with the mean reconstruction. Further, as one expects, the reconstruction quality degrades when the noise level is increased:

ISBN 978-3-527-33606-7

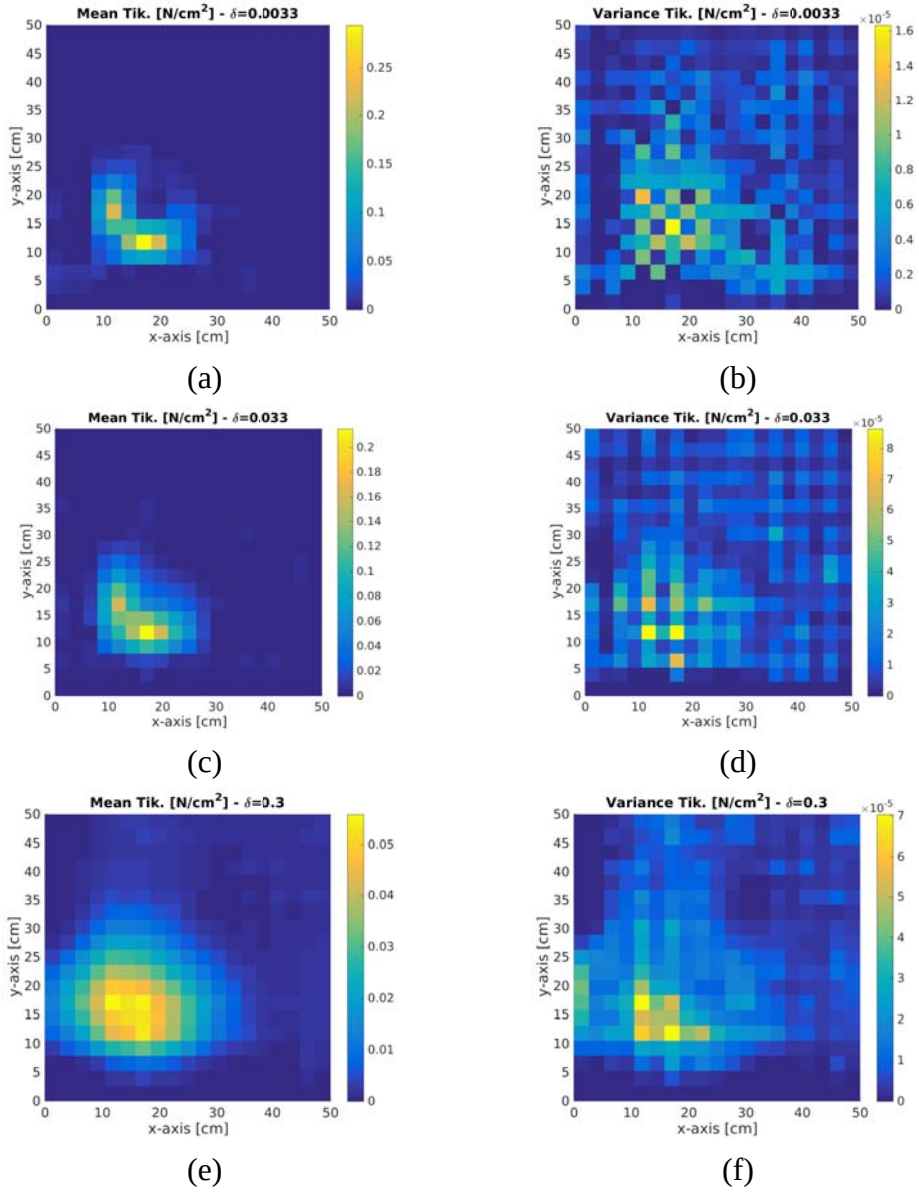


Fig. 5.37 *Reconstruction of f_1 by Tikhonov regularization with discrepancy principle for different relative noise levels. Reconstructions in $[\mathrm{N/cm^2}]$, units of x- and y-axes are cm. Noise levels $\delta = 0.0033$ (top row), $\delta = 0.033$ (mid row), and $\delta = 0.3$ (bottom row). Left column: Mean reconstruction of 100 reconstructions from independent noisy data sets. Right column: Variance of the 100 reconstructions.*

 ISBN 978-3-527-33606-7

For $\delta = 0.0033$, one notes for instance in Figure **5.37**(a) and Figure **5.38**(a) the corner due to the non-convex part of the original load in Figure **5.36**(c). When increasing the noise level to $\delta = 0.033$, this detail disappears and further increasing δ to 0.3, merely the location of the load is correctly identified. Further, the peak values of the reconstructions considerably decrease for higher noise levels due to a smearing effect caused by the higher degree of regularization required to ensure stability.

ISBN 978-3-527-33606-7

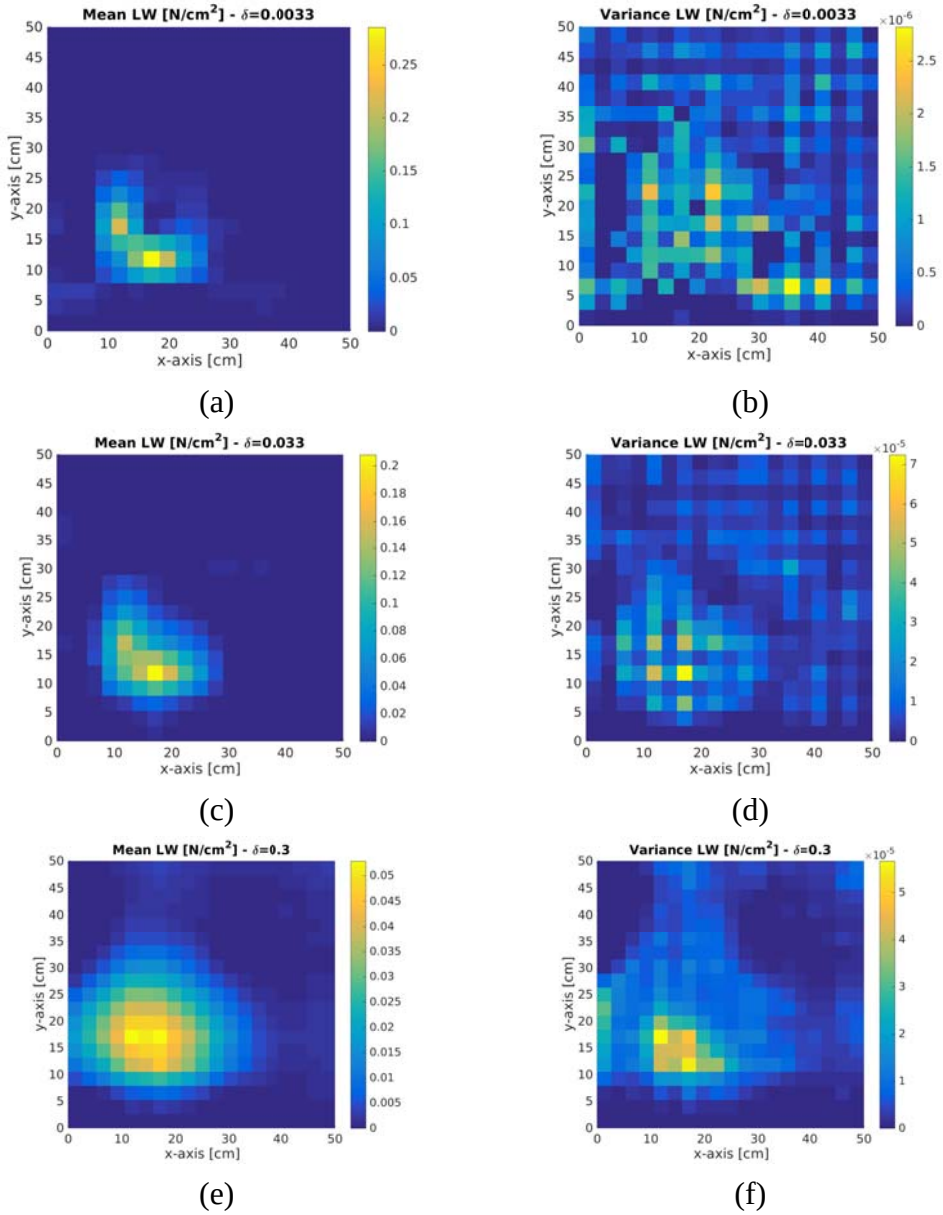


Fig. 5.38 *Reconstruction of f_1 by Landweber iteration with discrepancy principle for different relative noise levels. Reconstructions in $[\mathrm{N/cm^2}]$, units of x- and y-axes are cm. Noise levels $\delta = 0.0033$ (top row), $\delta = 0.033$ (mid row), and $\delta = 0.3$ (bottom row). Left column: Mean reconstruction of 100 reconstructions from independent noisy data sets. Right column: Variance of the 100 reconstructions.*

 ISBN 978-3-527-33606-7

Considering reconstructions of the second load f_2 from Figure **5.36**(d) in Figures **5.39** and **5.40**, the same conclusions on the reconstruction quality as for the load f_1 apply – variances of the reconstructions are generally rather small and the reconstruction quality decreases when the noise level increases.

ISBN 978-3-527-33606-7

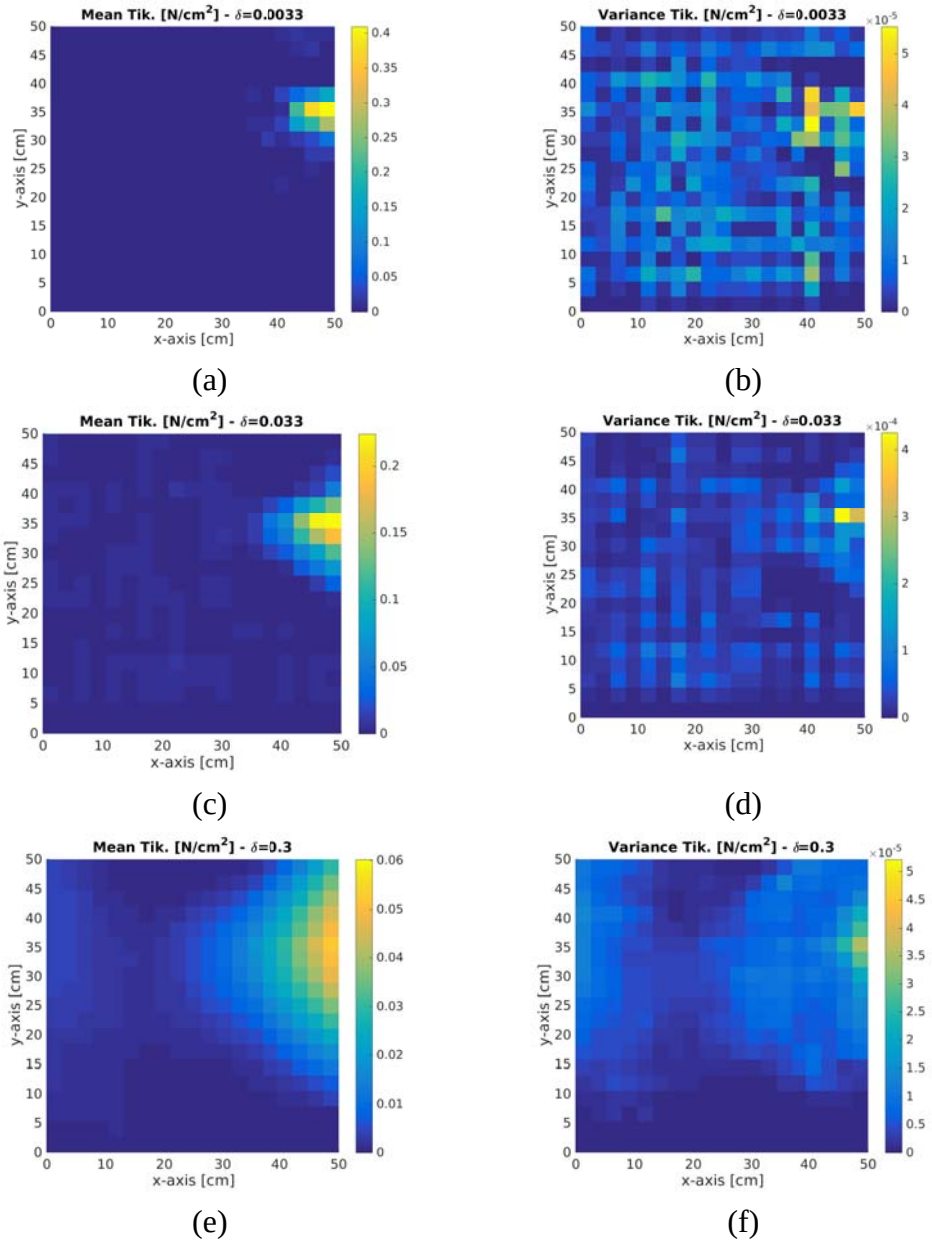


Fig. 5.39 *Reconstruction of f_2 by Tikhonov regularization with discrepancy principle for different relative noise levels. Reconstructions in [$\mathrm{N/cm^2}$], units of x- and y-axes are cm. Noise levels $\delta = 0.0033$ (top row), $\delta = 0.033$ (mid row), and $\delta = 0.3$ (bottom row). Left column: Mean reconstruction of 100 reconstructions from independent noisy data sets. Right column: Variance of the 100 reconstructions.*

 ISBN 978-3-527-33606-7

Let us finally compare computation times for Tikhonov regularization and the Landweber iteration, both using the discrepancy principle for determining the regularization parameter or the stopping index of the scheme, see (5.32) and (5.48). Tab. **5.2** shows that Landweber regularization should not be used for small noise levels: One iteration basically requires two matrix-vector multiplications such that the runtime essentially depends on the number of iterations required to satisfy the discrepancy principle. (There are far better iterative regularization methods for small noise level, such as the conjugate gradients method, see [BL14] or [EHN96].) As mentioned in the last section, this iteration number behaves like δ^{-2}, such that the Landweber iteration becomes infeasible for small noise level. On the other hand, for large noise levels 0.1 or 0.3, the Landweber iteration is an efficient regularization scheme, that keeps up with or even outperforms Tikhonov regularization in terms of speed and reconstruction quality. The runtime of Tikhonov regularization is governed by the sequence α_n from (5.43), as that sequence determines how often one has to compute solutions $p_{\alpha_n}^{\delta}$ to the linear system (5.37). Note, however, that the conclusion to decrease runtime by choosing a rapidly decreasing sequence is misleading, due to increasing instability.

Relative noise level	0.0033	0.01	0.033	0.1	0.3
Mean runtime Tikhonov regularisation	1.43	1.09	1.01	0.965	0.858
Variances	0.0541	0.0001	0.0001	0.0001	0.0004
Mean runtime Landweber iteration	62.8	18.6	5.09	1.06	0.178
Variances	17.4	0.739	0.238	0.0219	0.0012

Tab. 5.2 *Mean computation times and variances for Tikhonov and Landweber iteration combined with the discrepancy principle (computed on an 2.4GHz Intel Core 2 Duo processor).*

ISBN 978-3-527-33606-7

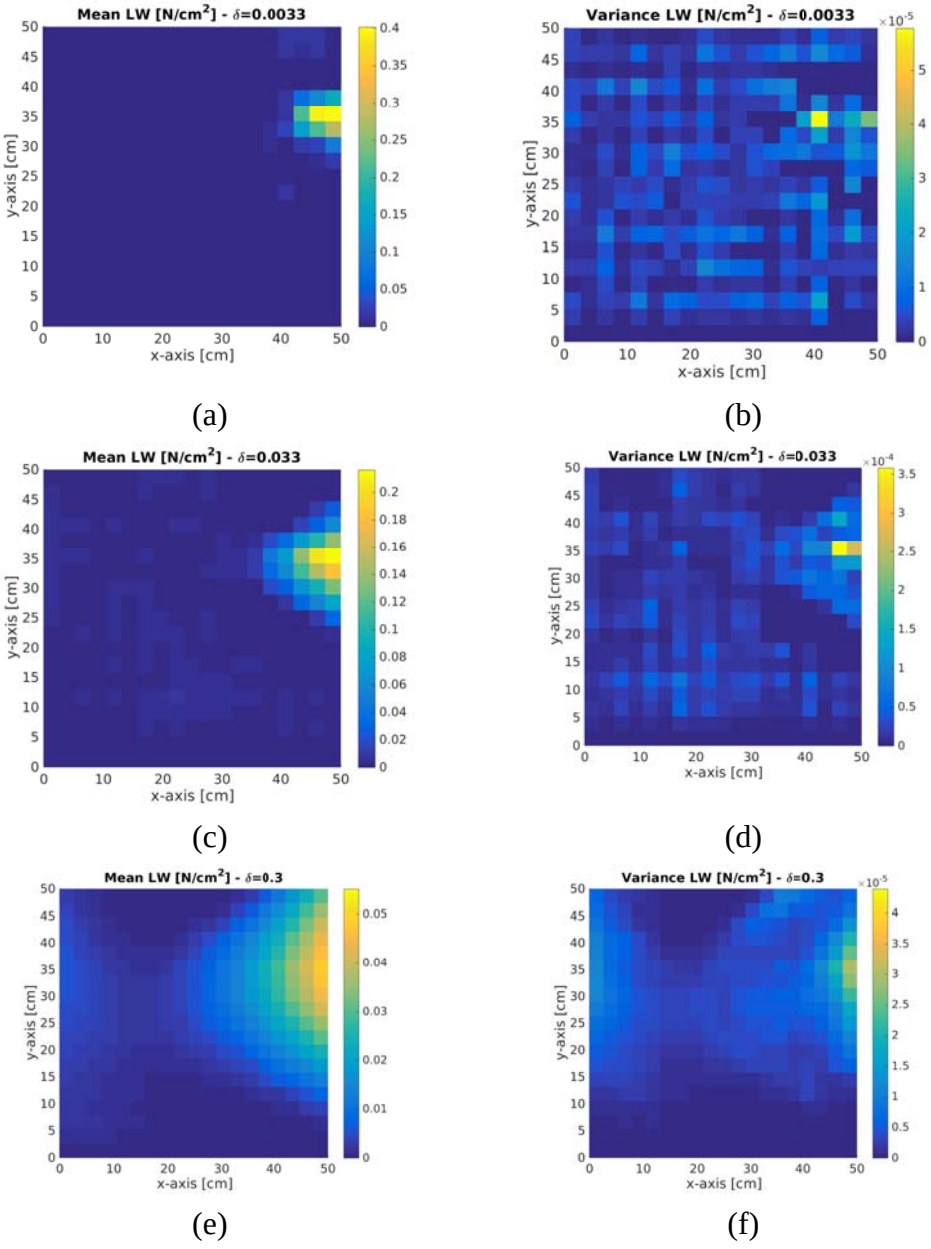


Fig. 5.40 *Reconstruction of f_2 by Landweber iteration with discrepancy principle for different relative noise levels. Reconstructions in $[\mathrm{N/cm^2}]$, units of x- and y-axes are cm. Noise levels $\delta = 0.0033$ (top row), $\delta = 0.033$ (mid row), and $\delta = 0.3$ (bottom row). Left column: Mean reconstruction of 100 reconstructions from independent noisy data sets. Right column: Variance of the 100 reconstructions.*

 ISBN 978-3-527-33606-7

5.4 The Unknown World: Model-free Computing and Machine Learning [Bosse]

5.4.1 Machine Learning - an Overview

An advantage of Machine Learning (ML) compared with numerical algorithms is the ability to handle problems with a Non-Polynomial (NP) complexity class **[WUE16]**, i.e., an exponential increase of computation time with respect to the data size. Basically learning algorithms can be used to improve a system behaviour either at run-time or at design-time (or a hybrid approach of both) by optimizing function parameters to increase mainly estimation accuracy. Machine learning is based either on example data (training with labeled data) or on past experience with reward feedback retrieved at run-time of a system. The fields of application range from the optimizing and fitting of parameters of evaluation functions to full feature extraction classifiers. Machine learning is often used if there is no or only an incomplete world model specifying on behavioural or functional level the output change of a module (the system or a part of it) in response of a change of the input stimulus.

A practically orientated classification of machine learning schemes (based on **[SAM11]**), suitable for information extraction in sensor networks and supporting model-free information retrieval is given by the following list:

Supervised Learning

A process that learns a model, basically a function mapping input data (of the past) to an output data set using data comprising examples with an already given mapping, i.e., using labeled data (see Fig. **5.41**). Two typical examples are classification and regression. Supervised learning produces a data base requiring a high demand of storage resources, but the resulting learned model can be considerable small (e.g., a decision tree).

Unsupervised Learning

This is generally a process which seeks to learn structure of data (see Fig. **5.41**) in the absence of an identified output (like in supervised learning with labeled data) or a feedback (reinforcement learning). Examples are clustering or self-organizing maps trying to group similar unlabeled data sets.

Semi-supervised Learning

This is the combination of supervised and unsupervised learning techniques creating a hybrid learning architecture (see Fig. **5.41**).

Reinforcement Learning

In some situations, the output of a system (the impact of the environment in that the system operates) is a sequence of actions, and the sequence of correct actions is important,

ISBN 978-3-527-33606-7

rather than one single action. Reinforcement learning seeks to learn a policy mapping states to actions that optimizes a received reward (the feedback), finally optimizing the behaviour of a system (the reactivity). It is different from data classifiers and relates closely to the autonomous agent behaviour model, discussed in Section **6.4**. There are no trained example situations for correct or incorrect behaviours, only rules evaluating and weighting actions and their impact on the environment and the system.

Association Learning

The goal of association learning is to find conditional probabilities of association rules between different items of data sets. An association rule has the form $X \rightarrow Y$, where X and Y are item sets. The association rule belongs to a conditional probability $P(Y|X)$ giving the probability that if X occurs, then set Y is also likely to occur. Giving user-specified support and confidence thresholds, the a-priori algorithm developed by **[AGR96]** can find all association rules between two sets X and Y. This proposed algorithm can be parallelized.

Classification

Classifier systems can be considered as modeling tools. Given a real system without a known underlying dynamics model, a classifier system can be used to generate a behaviour that matches the real system. The classifier offers rules-based model of the unknown system. There is a very large number of classification algorithms, for example, commonly used, decision trees (i.e., C4.5 algorithm), instance-based learners (i. e., nearest-neighbour (NN) methods), support vector machines (basically linear classifiers), rule-based learners, neural networks, and Bayesian networks. The main purposes of classification in sensor networks are knowledge extraction and pattern recognition. For example, in **[BOS13B]**, C4.5 and k-NN algorithms were used to classify a 18-dimensional sensor vector of a flat rubber plate to a (F,X) vector providing a strength classification of an applied load (F) with an estimation of the spatial position (X).

Decision trees as one outcome of a machine learning classifier have low computational resource requirements and can be implemented directly on microchip level, for example, used for energy management in **[BOS11B]**.

Clustering

Clustering is basically an unsupervised learning with the goal to group data sets sharing similar characteristics in clusters. The main goal is finding structure in the given set of data. A common clustering algorithm is the k-mean algorithm, which quantifies vectors and compute the distance between vector, finally grouping nearest vectors in cluster segments (see, e.g., **[BEL15]**). A Self-organizing map is another well known clustering algorithm.

Regression

Like classification, regression is a supervised learning approach. The goal is to learn an approximation of a real-valued function mapping an input data vector (real-valued input variables) to an output vector (the mean of response variables).

Supervised machine learning and *statistical pattern recognition* can be used to compute the response of a technical structure due to load or damage situations based only on the measured sensor data **[FAR13]**. Machine learning based *classification* can be used in the information levels 0 to 3 of Fig. **5.26**, but requires damage or load quantification to enable classification. Figueiredo et al. **[FIG10]** proposed machine learning methods for the damage detection under varying operational and environmental conditions using a hybrid approach with neural network algorithms. *Neural network algorithms* can be used in conjunction with self-organizing multi-agent systems and offering limited computational and data complexity promising microchip level implementations.

Unsupervised learning can be used for novelty detection. The algorithm simply indicates if the data come from a normal operating condition or not ("something is wrong"). *Regression* algorithms provide the output of several continuous variables, i.e., the diagnosis outputs the Cartesian coordinates of the fault, or the length of a crack. The regression problem can be nonlinear and is suitable for neural networks **[FAR13]**.

Machine learning is well suited for the analysis process and the feature extraction (i.e., the damage estimation of a technical structure) in SHM applications **[WOR07]**.

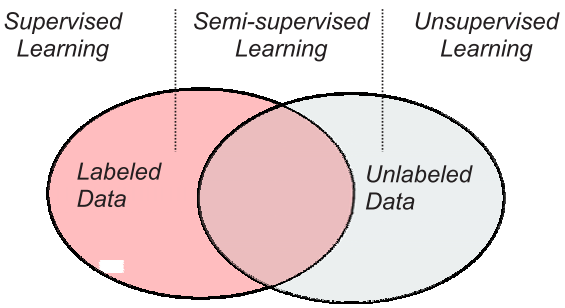


Fig. 5.41 *Transition from supervised to unsupervised learning*

5.4.2 Learning of Data Streams

Commonly, supervised machine learning consists of two phases: The Learning and the Analysis (predictive classification). The learning phase collects a number of experimental or simulated data sets assigned to a label vectors given by the supervisor, i.e., a training data set (X, y) is acquired off-line, mapping input data X on labels y. After the classifier was trained, real-time data can be processed, i.e., at run-time, the classifier is fed by data sampled at run-time. Incremental learning adds new training data sets (X', y') at run-time, trying to improve the classifier results, though additional training sets can degrade the classification results, too. The deployment of ML in Stream-based data processing

requires incremental learning machines. In sensing applications, the sensor data can be continuously over time and some kind of incremental and adaptive learning is required. There are different ML algorithms suitable for incremental learning, e.g., Decision Trees or Neural Networks, overlapping and interleaving learning and analysis phases (see Fig. **5.42** (b)).

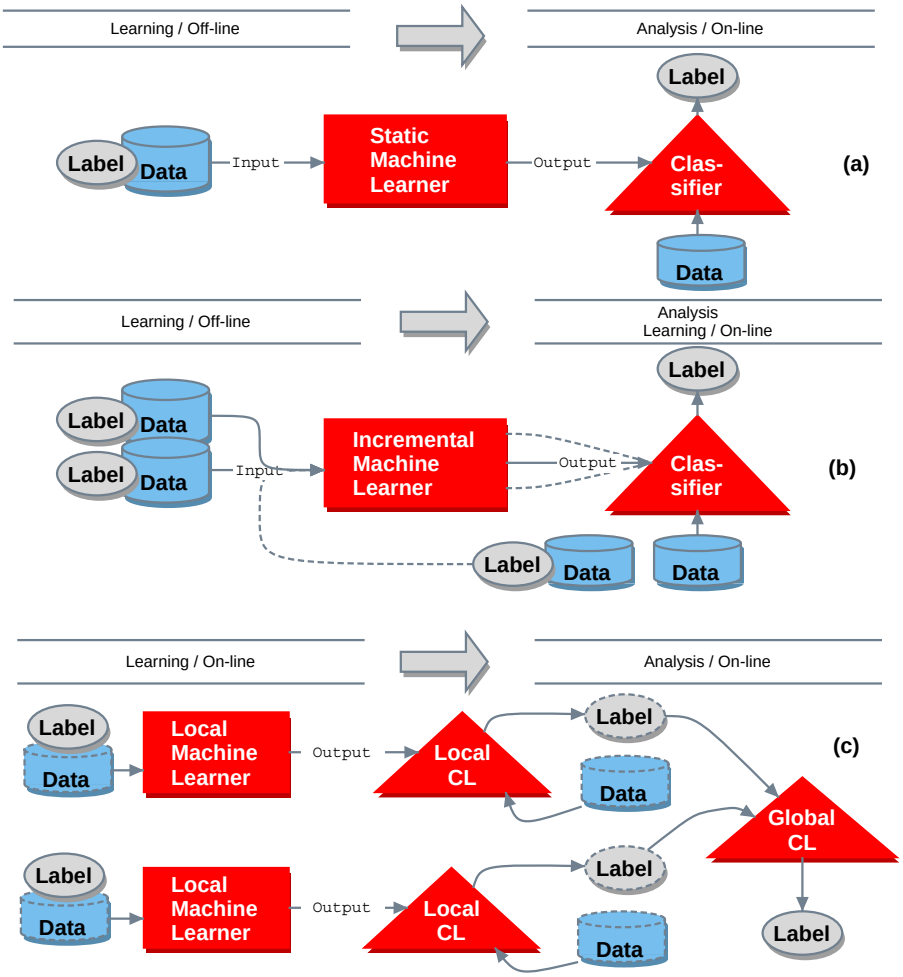


Fig. 5.42 *Comparison of static (a), dynamic incremental (b), and distributed (c) learning*

For example, Support Vector Machines (SVM) **[LAS06][RAL01]**, Neural Networks (NN), or Decision Tree Classifiers are used in incremental learning environments, combined with agent-based distributed learning **[CHO09]**. But SVMs are basically linear classifiers not suitable for high-dimensional data- and label vectors like they are present in sensing applications. It is important in incremental learning that the learner not relies on

the storage of old data sets already learned with a full re-computation, i.e., the learning algorithm must avoid re-learning of the entire data set each time an iteration step is performed. Decision tree (DT) and rule (DR) learners seem to be the most promising ML methods for real-time stream-based learning. They can be learned on sequential scans. DTs are robust to errors and drift in the data, though some effort is required to perform incremental learning efficiently with low computational power using, e.g., Very Fast Decision Tree (VFDT) learning applied to data streams **[GAM11]**.

5.4.3 Learning with Noise

One major issue in learning a classification model or performing clustering is the uncertainty of sensor data, i.e., the random noise supposed to the original sensor data. Noise is always present, related either to a statistical variation (based on a physical effect) or being sporadic based mostly on failures of the sensor or electronics. Noise can significantly reduce the prediction quality and accuracy of a learned model. Different learning algorithms have a different sensitivity to noise. Fuzzy learning by adding uncertainty intervals to data can improve the classification stability significantly, shown in Sec. **5.4.5**.

5.4.4 Distributed Event-based Learning

Distributed learning divides a spatial distributed data set in clustered regions and applies learning to the limited local regions, based on a divide and conquer approach.I.e., there are multiple learning instances learning a local model (using spatially bounded data) that is finally merged in a global model. Event-based learning activates learners if there is a local stimulus detected. This is shown in Fig. **5.42**(c). Decision trees are simple models derived from learning with training data, and well suited for agent-based learning. A learned model is used to map data set vectors on class values. The tree consists of nodes testing a specific attribute variable, i.e., a particular sensor value, creating a path to the leaves of the tree containing the classification result, e.g., the load situation class. Among the distribution of the entire learning problem, event-based activation of learning entities can improve the system efficiency significantly. Commonly the locally sampled sensor values are used for an event prediction.

The event-based regional learning leads to a set of classification results, which can differ significantly, i.e., the classification set can contain wrong predictions. To filter out and suppress these wrong predictions, a global major vote election is applied. All nodes having performed a regional classification send their result to the network collecting all votes and perform an election. This election result is finally used for the global prediction. The variance of different votes can be an indicator for the trust of the election giving the right prediction.

The following Eq. **5.55** summarizes the spatial distribution of learning. The global model learner M is divided in multiple local learner m, computing a local model k based on local data d, a tuple set consisting of input data s (e.g., sensor data) and classification

ISBN 978-3-527-33606-7

labels *l*. The local classification function *k* is applied to unknown local data *s* predicting a specific label *l*, finally merged in a global prediction.

$$\begin{aligned} & M : D \to K(S) && m_{i,j} : d_{i,j} \to k_{i,j}(s) \\ & l \in \boldsymbol{L} && k_{i,j} : s_{i,j} \to l_{i,j} \\ & K : S \to l \quad \xrightarrow{\text{Distribution}} && K : (l_{1,1}, l_{1,2}, \cdots) \to l \\ & D : \{(S^1, l^1), (S^2, l^2), \cdots\} && d_{i,j} : \{(s_{i,j}^1, l^1), (s_{i,j}^2, l^2), \cdots\} \\ & S : \begin{pmatrix} x_{1,1} & \cdots & x_{n,1} \\ \vdots & \ddots & \vdots \\ x_{1,m} & \cdots & x_{n,m} \end{pmatrix} && s_{i,j} : \begin{pmatrix} x_{i-u,j-v} & \cdots & x_{i+u,j-v} \\ \vdots & \ddots & \vdots \\ x_{i+u,j-v} & \cdots & x_{i+u,j+v} \end{pmatrix} \end{aligned} \tag{5.55}$$

5.4.5 ε-Interval and Nearest-Neighbourhood Decision Tree Learning

Traditional Decision Tree Learner (DTL) (e.g., using the *C4.5* algorithm) select data set attributes (feature variables) for decision making only based on information-theoretic entropy calculation to determine the impurity of training set columns (i.e., the gain), which is well suited for non-metric symbolic attribute values, like color names, shapes, and so on. The distinction probability of two different symbols is usually 1. Numerical sensor data is noisy and underlies variations due to the measuring process and the physical world. Two numeric (sensor) values *a* and *b* have only a high distinction probability if the uncertainty intervals [*a*-σ,*a*+σ] and [*b*-σ,*b*+σ] due not overlap. That means, not only the entropy of a data set column is relevant for numerical data, the standard deviation σ and value spreading of a specific column must be considered, too. To improve attribute selection for optimized data set splitting, a column ε-interval entropy computation was introduced, that extends each value of a column vector with an uncertainty interval [v_i-ε,v_i+ε]. Values with overlapping intervals are considered to be non distinguishable, lowering the entropy *entropy* (with ⌊x: lower bound of a value/interval, ⌈x: upper bound, and |v| as the size of a vector), with the computation given by Eq. **5.56**.

$$
\begin{aligned}
&\text{entropy}_{\varepsilon}(cols,\varepsilon) = \sum_{i=1..|cols|} -\text{prob}_{\varepsilon}(col_i, cols, \varepsilon)\log_2(\text{prob}_{\varepsilon}(col_i, cols, \varepsilon)) \\
&\text{prob}_{\varepsilon}(v, cols, \varepsilon) = \frac{\sum_{i=1..|cols|} \begin{cases} 0 : \text{overlap}([cols_i - \varepsilon, cols_i + \varepsilon], [v-\varepsilon, v+\varepsilon]) \\ 1 : otherwise \end{cases}}{|cols|} \\
&\text{overlap}(v_1, v_2) = \begin{cases} true : (\lceil v_1 \geq \lfloor v_2 \wedge \lceil v_1 \leq \lceil v_2 \rceil \vee \\ \qquad (\lceil v_2 \geq \lfloor v_1 \wedge \lceil v_2 \leq \lceil v_1 \rceil \\ false : otherwise \end{cases} \\
&\text{distance}(v_1, v_2) = \left| \frac{\lfloor v_1 + \lceil v_1}{2} - \frac{\lfloor v_2 + \lceil v_2}{2} \right|
\end{aligned}
\tag{5.56}
$$

The ε-entropy is calculated for all data set columns, and the attribute (feature variable) for the column with highest entropy value is selected. The column can still contain non-distinguishable values with overlapping 2ε intervals. All overlapping 2ε values are grouped in partitions that cannot be classified (separated) by the currently selected attribute variable. Only partitions - ideally containing only one data set value - are used for a classification selection. All data sets in one partition create a sub-tree of the current decision tree node. If there is only one partition available (containing more than one class target, a data set attribute selection is based on the column with the highest standard deviation, but the 2ε separation cannot be guaranteed in this case, lowering the prediction accuracy. The basic principle of the learning algorithm, which is an adaptation of a common discrete C4.5 Decision Tree Learner, is shown in Alg **5.4**. It creates a model with attribute value interval selection, e.g. x∈[500..540], instead the commonly used and simplified relational value selection, e.g., x < 540, which is an inadmissible extrapolation beyond the training set boundaries and prevent recognizing totally non-matching data.

Alg. 5.4 *Principle Learning and Classification algorithms. The entropy computation applying the 2ε interval to values is shown in Eq.* ***5.56****.*

```
type value = number | number range
The learned model is a decision tree with nodes and leaves
type model = Result (name: string) |
             Feature (name:string, featvals: model array) |
             Feature Value(val: value, child: model)

function createTree(datasets, target, features)
  1. Select all columns in the data set array with the target key
  2. If there is only one column, return a result leaf node with the target
  3. Determine the best features by applying entropy and value deviation
     computation
  4. Select the best feature by maximal entropy
  5. Cretae partitions from all possible column values for this feature
```

```
  6. If there is only one partition holding all values, go to step 10
  7. For each partition create a child feature value node
  8. For each child node apply the createTree function with the
     remaining reduced data set by filtering all data rows containing at least
     one value of the partition in the respective feature column of the data set,
     and by using a reduced remaining feature set w/o the current feature
  9. Return a feature node with previously created feature value child nodes.
     Finished.

  10. Select the best feature by maximal value deviation
  11. For each possible value create a feature value node
  12. For each child node apply the createTree function with the
      remaining reduced data set by filtering all data rows containing at least
      one value of the partition in the respective feature column of the data
      set, and by using a reduced remaining feature set w/o the current feature
  13. Return a feature node with previously created feature value child nodes.
      Finished.
end

function classify (model,dataset)
  I. Iterate the model tree until a result leaf is found.
  II. Evaluate a feature node by finding the best matching feature value node
      for the current feature attribute by finding the feature value with
      minimal distance to the current sample value from the data set.
end
```

The prediction (analysis and classification) algorithm is a hybrid approach, too. It consists of the tree iteration, but uses a simple nearest-neighbourhood estimation for selecting the best matching feature value with a given sample sensor value.

The learned DT is composed of result leaves and feature/feature value selection nodes. A learned DT model can be easily transformed in a table representation, enabling the implementation of learned DT models on hardware level using simple linear Look-up Tables (LUT), illustrated in Fig. **5.43**.

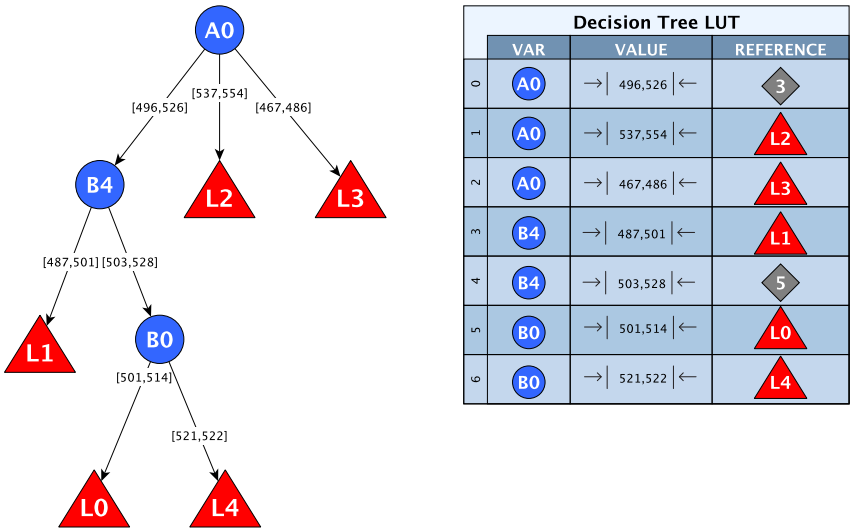


Fig. 5.43 *A learned decision tree and its linear look-up table representation (A_i: Sensor A on node i, B_i: Sensor B on node i, L_i: Load case i)*

5.4.6 Machine Learning - a Sensorial Material Demonstrator

A first attempt to investigate new smart materials was performed with experiments using a functional mock-up consisting of a flat rubber plate (equipped with nine bi-axial strain-gauge sensors and the sensor network previously introduced, 70 mm sensor distance) and an experimental set-up shown in Figure **5.44** with circular weights.

Figures **5.44** and **5.45** show the analysis of the difference between measured and predicted load positions (position accuracy) retrieved by machine learning with two different sensor array configurations. The plots show the spatial vector difference between the predicted and observed position with a mean value below 25 mm and 50 mm, respectively. The training set consisted of two different masses (103 g and 306 g) and 150 positions. During learning mode the position (of the weight) was monitored with a camera mounted above the rubber plate together with acquired sensor data delivered by the sensor network.

ISBN 978-3-527-33606-7

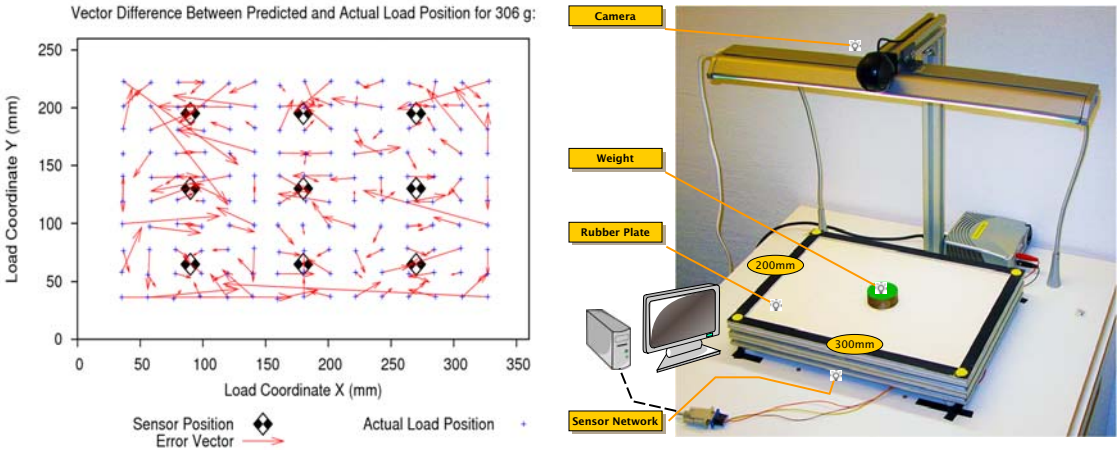


Fig. 5.44 *Experimental results of predicted load positions (306 g weight) with nine strain-gauge sensor pairs mounted on backside of a rubber plate (experimental test set-up shown on right side)* ***[PAN11B]****.*

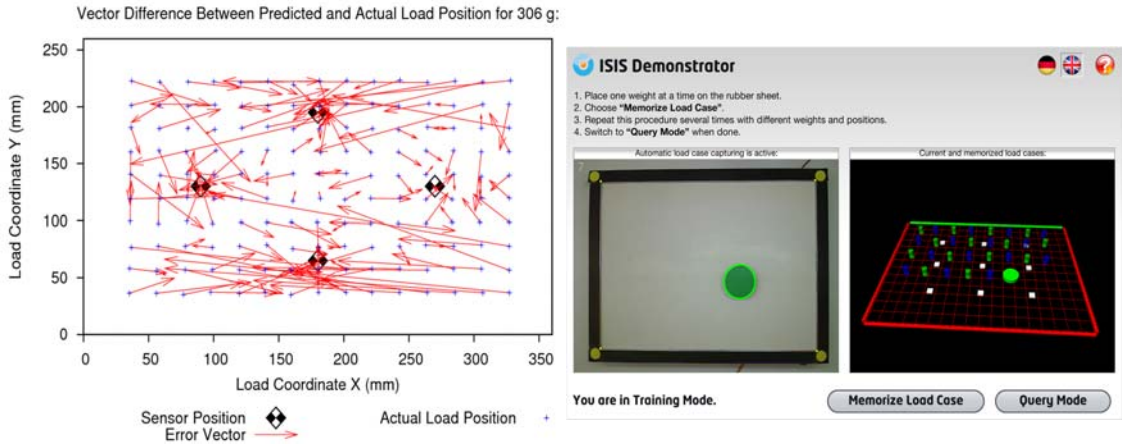


Fig. 5.45 *Experimental results of predicted load positions (306 g weight) with only four strain-gauge sensor pairs and ML graphical user interface (GUI, right side)* ***[PAN11B]****.*

A system with reduced number of sensors (Figure **5.45**, 150 mm sensor distance) results in a decrease of prediction position accuracy in the boundary region, but is still usable for structure load monitoring especially in the middle area spawned by the edges of sensors.

The sensor network attached below the rubber plate consisted of nine single sensor nodes, each equipped with Analogue-Digital conversions and digital processing units

(FPGA). Each sensor node was attached to one bi-axial aligned strain-gauge sensor pair. The sensor nodes were arranged in two-dimensional network, shown in Fig. **5.46**.

The sensor data was collected periodically from the network of smart sensor nodes, which already performed the signal pre-processing. The sensor data was finally processed by a computer to derive load information (interpolated position and load strength classification) by using supervised machine learning. Different machine learning methods were investigated, showing different quality and stability results of the predicted loads, and differing in the required number of training experiments to reach a certain level of quality and robustness **[PAN11B]**:

- k-Nearest-Neighbour ➠ Numerical Regression of the load position, mass, and displacement vectors;
- C4.5 Decision Trees ➠ Mass Classification;
- Neural Networks ➠ Numerical Regression of load position and mass.

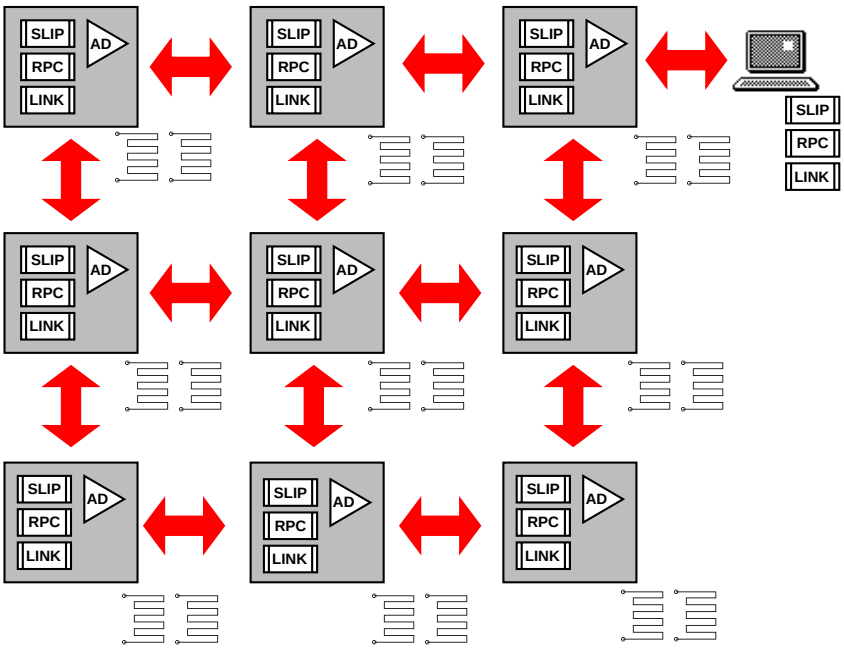


Fig. 5.46 *Two-dimensional Sensor Network with autonomous sensor nodes attached to a computer performing advanced data analysis using machine learning. Each smart sensor node consists of signal & digital data processing, and communication.*

Finally, the k-NN classificator showed the best overall results. The original proposed approach included FEM simulation to enable simulated virtual training of the ML classi-

ISBN 978-3-527-33606-7

fier, finally used in the load classification with data from the real sensor network sampled at run-time, can be found in **[BOS11C]**.

Messaging was performed in the sensor network by using an adaptive Δ-Distance-Routing Protocol (SLIP) based on backtracking, which will be discussed in Sec. **6.2.7**. This adaptive routing scheme enables the delivery of messages even in the case of incomplete (missing communication connections) and irregular (missing or down nodes in the 2-dimensional mesh grid) networks.

5.5 Robustness and Data Fusion [Bosse]

Reliability is a key factor in the design and the operation of distributed sensor networks integrated in materials and technical structures, which in current works is often not considered in a rigorous way on system design level. Failures in DSN can be categorized in:

1. Sensor failures
2. Communication link failures
3. Data processing failures with different sub-levels
4. Algorithmic design errors
5. Race conditions, dead locks, and live-locks in concurrent systems

This study is not concerned with the various sources of physical failure, nor their relative importance. Instead, we will discuss potential remedies in case of failure as well as measures that help maintain functionality even after failure has occurred.

Process failures (of the operational units or the sensor node itself) can be arbitrary (i.e., due to electronic failures), part of a crash with possible recovery, a crash without recovery, or related to omissions (not all operations of a process were performed successfully or finished) **[GUE06]**. Arbitrary faults are the most difficult to handle, and can be a result of a wrong design on algorithmic (programming) level. Omission faults are mainly related to incomplete communication (e.g. a process is not responding to a communication request). In contrast process crashes can be detected and "healed" with a process recovery, either restoring the state of the process (consisting of a control and data state) to an older backup state, or just by restarting the process (restoring the initial state).

Communication failures and robustness aspects are discussed in Sec. **6.2.8**.

There are different approaches to achieve programming fault tolerance. Basically there are forward and backward error recovery (assuming the correct operation of the host PE of a process). In backward recovery strategies an older backup process state (consisting of the data and control state) is restored if an error occurs. Forward recovery tries to correct the wrong behaviour of the process more precisely by identifying the error and modifying the system state containing the error. Exception handling provided on programming level by high-level software languages like Java, JavaScript, ML, or ADA, or by high-level hardware SoC design languages like ConPro **[BOS11A]** can help to handle and to detect runtime errors in a deterministic and efficient way.

If errors cannot be safely identified, which is always the case in defective PEs, voting based approaches can be used. Redundancy can be provided by executing the same computation by different PEs (or at least processes) and performing a major voting. N-version programming is a similar but more powerful and robust approach. In this approach the computation is performed by different algorithms executed in different processes with a

ISBN 978-3-527-33606-7

final major voting **[WU99]**. But redundant computation always increases the resource demands significantly (for storage, operations, and communication).

Communication failures can be compensated with network topologies providing alternatives for path routing leading to path redundancy, discussed in Section **6.2.2**.

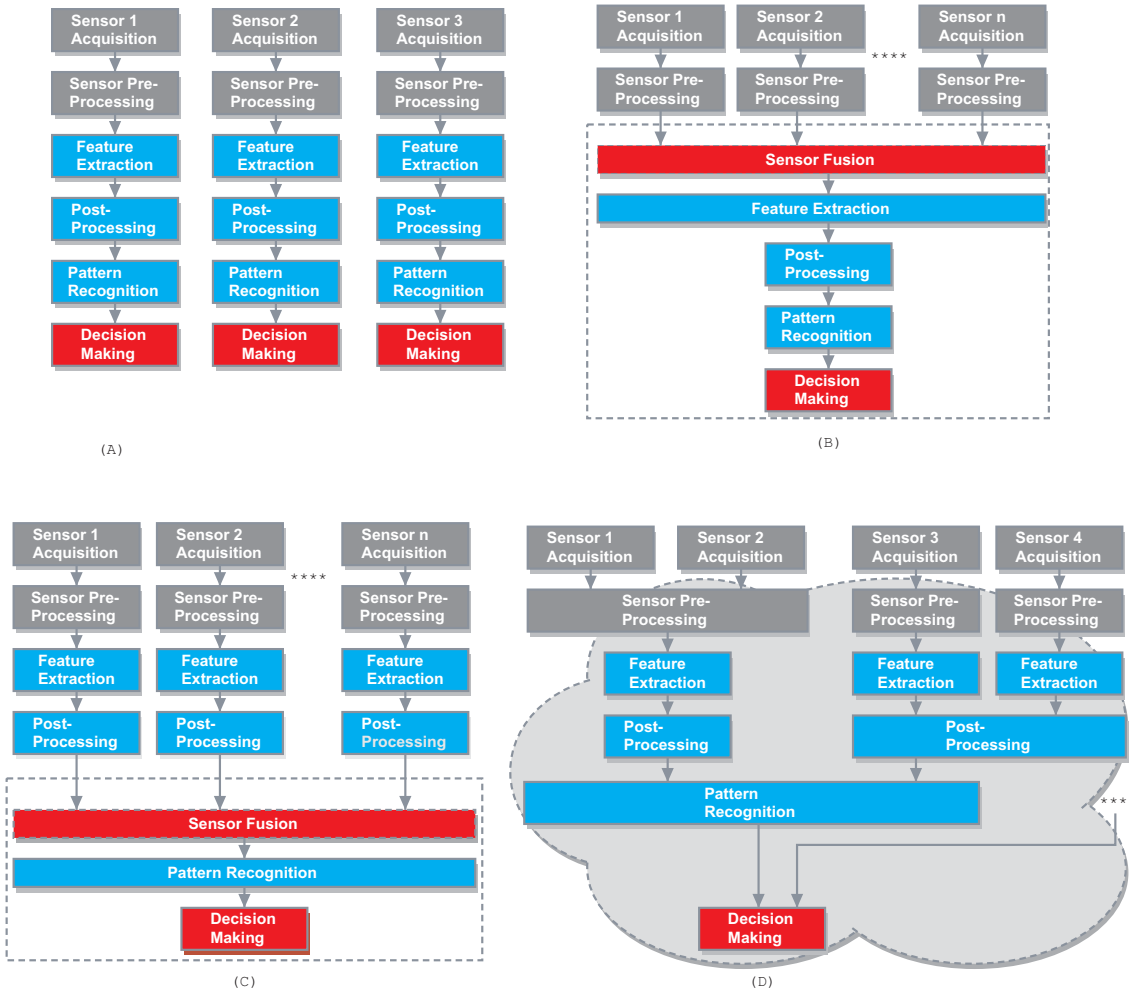


Fig. 5.47 *Different strategies in sensor data fusion and fusion architectures (Adapted from **[FAR13]**)*
(A) Single Sensor Processing (B) Centralized Sensor Fusion (C) Distributed Sensor Fusion (D) Heterogeneous and Hierarchical Sensor Fusion Architecture

Sensor fusion (also known as multi-sensor data fusion) is another well known technique to increase the robustness and confidence of a sensor signal processing system by using

multiple sensor outputs to improve a decision making process. Sensor fusion can be used to:

- Improve the overall signal-to-noise ratio;
- Increase the robustness and reliability;
- Extend the coverage to supply a more complete picture of the environment and system under perception;
- Improve the resolution of a measurement;
- Increase the confidence of sensor information and decision making processes.
- Reduce the measurement time.

It is a process of combining observations from different sensors. A multi-sensor data fusion system can be seen as a distributed system of autonomous modules, well fitting in the concept of distributed sensor networks.

The fusion process can be classified in different levels **[FAR13]**:

- Raw sensor-level fusion before any pre-processing;
- Feature-level fusion after sensor signal pre-processing;
- Pattern-level fusion combining feature vectors;
- Decision-level fusion combining multiple decisions or classifications.

There are different fusion architectures, shown in Fig. **5.47**. The single sensor processing architecture (A) has one fully expanded processing chain for each sensor, finally producing multiple results. Each processing chain consists of a sensor pre-processing stage, feature extraction (or information computation), post processing, pattern recognition, and finally the decision making stage. A centralized sensor fusion architecture (B) has multiple sensor pre-processing chains combined in a centralized fusion stage (Raw sensor-level fusion). A distributed sensor fusion architecture (C) performs the sensor pre-processing, feature extraction, and the post-processing distributed, and finally the post-processed data is combined in a centralized fusion stage. A heterogeneous and hierarchical architecture (D) is composed of a tree with multiple distributed processing and combination stages with one centralized decision making stage.

5.5.1 Robust System Design on System Level

Serial and parallel system composition are the most common basic structures in data processing system design, which can be applied on software and hardware architecture level, illustrated in Fig. **5.48**. Data processing nodes in a mesh-like sensor network can be considered as a parallel composition. Furthermore, parallel data processing on microchip level can be easily exploited by using multi Register-Transfer architectures **[BOS11A]**. On

ISBN 978-3-527-33606-7

the other hand a chain or bus network topology is a serial composition of initially independent systems. The functional composition of programs and algorithms leads to a serial system behaviour, too, without providing any robustness against partial failures.

The reliability R (probability to operate without a failure a certain time t) and the mean time to failure (*MTTF*) of such a serial system compared with a parallel system is given by Eq. **5.57**a/b) (λ is the failure rate of a module, one part of the system)**[KOR07]**.

$$R_{serial}(t) = \prod_{i=1}^{N} R_i(t),\ \ MTTF_{serial} = \frac{1}{\sum \lambda_i} = \frac{1}{\lambda_s} \qquad (a)$$

$$R_{parallel}(t) = 1 - \prod_{i=1}^{N} (1 - R_i(t)),\ \ MTTF_{parallel} = \sum \frac{1}{k\lambda_i} \qquad (b) \qquad (5.57)$$

$$R_{serial+parallel}(t) \le 1 - \prod_{i=1}^{N} (1 - R_{path,i}(t)) \qquad (c)$$

This means the failure of a serial system is an accumulation of all independent failures R_i of one module with a resulting constant failure λ_s rate, whereas a parallel system is a cumulation of liveliness, with a non-constant failure rate decreasing with each failure of a part. The reliability (or *MTTF*) of mixed systems consisting of weaved serial and parallel structures are difficult to calculate. An upper bound (Eq. **5.57**c) of the overall system reliability can be given, assuming a decomposition of the system in serial paths operating parallel (and independently) with a single reliability R_{path}. This implies that in a mixed system some modules can be removed without compromising the overall system operation.

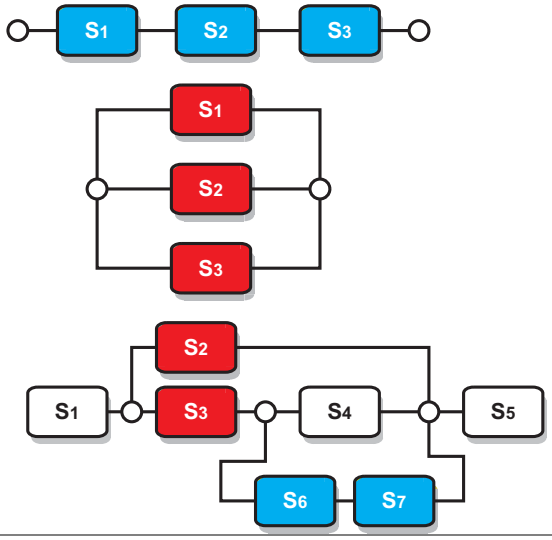


Fig. 5.48 *Serial, parallel, and mixed system designs*

M-of-N Systems

Majority voting can be used to isolate modules producing faulty results, which can be implemented with agents (see Sec. **6.4**). A major voting assumes that at least M of N independent modules, performing the same computation, are working properly. This technique offers fault tolerance by redundancy. If $R(t)$ is the reliability of one independent module (the probability to being operational at time t), then the reliability of the whole system is (Eq.**5.57**a) **[KOR07]**:

$$R_{M-of-N}(t) = \sum_{i=M}^{N} \binom{N}{i} R^i(t)\left(1-R(t)\right)^{N-i}$$
$$R^{corr}_{M-of-N}(t) = (1-Q_{corr})R_{M-of-N}(t) \qquad (5.58)$$

If there is even a slight correlation Q (e.g., a probability for a common fault) between the individual modules then the overall system reliability decreases dramatically (Eq. **5.57**b), forcing a proper system design and the identification of correlation effects, for example, practically speaking, in the case of sensor networks there are shared resources like power supplies or communication structures which can fail! A wrong computation result must be considered as a partial fault of a module.

In addition to the previously described static redundancy based on voting and employing a large number of independent modules at the same time, dynamic redundancy which enable spare modules in the presence of detected faults of one actually active module can improve system efficiency (power, communication, utilization of processing elements).

ISBN 978-3-527-33606-7

Dynamic redundancy is obtained by using fault detection and reconfiguration. If there is only one active module at the same time, the estimation and detection of failures or incorrect result is critical, decreasing the system reliability. The system reliability can be increased by using a hybrid approach of voting (static redundancy) and fault detection (dynamic redundancy) **[KOR07]**.

Autonomous agent systems can reduce computational correlation effects on system level significantly. Replication and reconfiguration features of agents can offer efficient and dynamic M-of-N system implementations.

N-Version Systems

These can be considered as a more general approach of M-of-N system designs. In a N-version system the same computation is performed by different algorithms or implementations of algorithms, increasing the independence of parallel computing modules significantly. This approach covers programming and implementation errors at design time, too, in contrast to a M-of-N system with identical modules, which all can have the same design error and producing faulty results under some (rare) circumstances, basically depending on the input data. Version independence is important, otherwise correlated data processing errors can decrease the system reliability significantly (as already discussed with M-of-N systems).

Distributed Neighbourhood Detection

Failures of single sensors are hard to distinguish from random measuring errors. A well known technique is data fusion of different independent sensor signals detected in a bounded neighbourhood region around a center node, assuming nearby sensors measuring similar signals and the measurements are stochastically non-correlated. This technique can improve fault tolerance by detecting defective modules (sensors) and can increase the quality-of-service for signal monitoring significantly, as suggested in **[LUO06]**.

Dynamic Self-organizing Systems

In contrast to computational and communication robustness related often to a particular sub.-system or sensor node, the liveliness of the entire systems in presence of single and multiple failures on different levels is more important. The system liveliness is mostly defined by the goals of a a system, rarely on the state of components composing the entire system. For example, it is pointless if there is a node or senor failure in a spatially distributed strain gauge network if this not effects the overall load computation by using a numeric or machine learning mapping functions. But it is difficult to predict the overal system behaviour from particular failures. Due to this difficulties, Self-organizing Systems (SoS), commonly using multi-agent systems, can overcome this lack of a complete failure model. A SoS is characterized by some degree of autonomy of individual parts, adaptabil-

ity, and reconfiguration of system and sub-system configurations, for example, proposed in **[BOS15A]** using an agent-based SoS for feature detection in a load monitoring network.

ISBN 978-3-527-33606-7

5.6 References

[AGR96] Agrawal, R., & Shafer, J. C. (1996). Parallel mining of association rules. Ieee Trans. On Knowledge And Data Engineering, 8(6), 962–969

[ARM66] L. Armijo. Minimization of function having Lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16:1–13, 1966.

[AUB00] J.-P. Aubin. *Applied Functional Analysis*. Wiley, 2. edition, 2000.

[BAL06] Balageas, D., Fritzen, C., & Güemes, A. (Ed.). *Structural health monitoring*, ISTE, 2006

[Bak92] A.B. Bakushinskii. The problem of the convergence of the iteratively regularized Gauss–Newton method. *Comput. Maths. Math. Phys.*, 32:1353–1359, 1992.

[BAK10] Baker, R. J.: CMOS Mixed-Signal Circuit Design. 3rd Edition. John Wiley &Sons, Inc. and IEEE Press, 2010

[BB88] J. Barzilai and J. M. Borwein. Two-point step size gradient methods. *IMA J. Numer. Anal.*, 8:141–148, 1988.

[BEL15] J. Bell, Machine Learning - Hands-On for Developers and Technical Professionals. John Wiley & Sons, Ltd, 2015.

[BOL09] C. Boller, *Structural Health Monitoring - An Introduction and Definition*, in Encyclopedia of Structural Health Monitoring, Wiley, 2009

[BOS11A] S. Bosse, H*ardware-Software-Co-Design of Parallel and Distributed Systems Using a unique Behavioural Programming and Multi-Process Model with High-Level Synthesis*, Proceedings of the SPIE Microtechnologies 2011 Conference, 18.4.-20.4.2011, Prague, Session EMT 102 VLSI Circuits and Systems

[BOS11B] Stefan Bosse, Thomas Behrmann, *Smart Energy Management and Low-Power Design of Sensor and Actuator Nodes on Algorithmic Level for Self-Powered Sensorial Materials and Robotics*, Proceedings of the SPIE Microtechnologies 2011 Conference, 18.4.-20.4.2011, Prague, Session EMT 101 Smart Sensors, Actuators and MEMS, 2011, DOI:10.1117/12.888124

[BOS11C] F. Pantke, S. Bosse, D. Lehmhus, M. Lawo, *An Artificial Intelligence Approach Towards Sensorial Materials*, Proceedings of the Future Computing 2011 Conference, DOI 10.13140/2.1.3124.0647

[BOS12A] S. Bosse, F. Pantke, *Distributed computing and reliable communication in sensor networks using multi-agent systems*, Prod. Eng. Res. Devel., 2012, DOI 10.1007/s11740-012-0420-8

[BOS12B] S. Bosse, F. Pantke, and F. Kirchner, *Distributed Computing in Sensor Networks Using Multi-Agent Systems and Code Morphing*, ICAISC Conference, Poland, Zakapone, 2012

[BOS13B] S. Bosse, F. Pantke, S. Edelkamp, *Robot Manipulator with emergent Behaviour supported by a Smart Sensorial Material and Agent Systems*, Proceedings of the Smart Systems Integration 2013, Amsterdam, 13.3. - 14.3.2013, NL

[BOS15A] S. Bosse, Design and Simulation of Material-Integrated Distributed Sensor Processing with a Code-Based Agent Platform and Mobile Multi-Agent Systems (Article), Sensors, 15 (2), pp. 4513-4549, 2015, DOI:10.3390/s150204513

[BOS16A] S. Bosse, A. Lechleiter, *A hybrid approach for Structural Monitoring with self-organizing multi-agent systems and inverse numerical methods in material-embedded sensor networks*, Mechatronics, 2015, doi:10.1016/j.mechatronics.2015.08.005, in press

[BL14] S. Bosse and A. Lechleiter. Structural health and load monitoring with material-embedded sensor networks and self-organizing multi-agent systems. *Procedia Technology*, 15:669–691, 2014.

[BNS97] B. Blaschke, A. Neubauer, and O. Scherzer. On convergence rates for the iteratively regularized Gauss-Newton method. *IMA Journal of Numerical Analysis*, 17:421–436, 1997.

[CAN10] Cannata, G., Dahiya, R., Maggiali, M., Mastrogiovanni, F., Metta, G., & Valle, M. (2010). Modular Skin for Humanoid Robot Systems. CogSys 2010 Conference Proceedings (Vol. 231500

[CAR04] Carden, E. P. (2004). *Vibration Based Condition Monitoring: A Review*. Structural Health Monitoring, 3(4), 355–377. doi:10.1177/1475921704047500

[CHO09] J. Choi, S. Oh, and R. Horowitz, "Distributed learning and cooperative control for multi-agent systems," Automatica, vol. 45, no. 12, pp. 2802–2814, Dec. 2009.

[CHO99] K. P. Chong, Health Monitoring of Civil Structures. Journal of Intelligent Materials Systems and Structures, Volume 9, Issue 11, 1999, pp. 892-898

[CHU11A] Chuck Moore et al., PB002 GreenArrays Architecture, 2011, http://www.greenarraychips.com

[CHU11B] Chuck Moore et al., WP002 GreenArrays Energy Conservation, 2010, http://www.greenarraychips.com

[CIA88] P. G. Ciarlet. Mathematical Elasticity. Vol. I: Three-Dimensional Elasticity. North-Holland, Amsterdam, 1988.

[DAH07] Dahiya, R., & Valle, M. (2007). Tactile sensing arrays for humanoid robots. Research in Microelectronics and Electronics Conference, 2007. PRIME 2007 (pp. 201–204)

[DAH10] Dahiya, R. S., Lorenzelli, L., Metta, G., & Valle, M. (2010). *POSFET devices based tactile sensing arrays*. Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 893–896. doi:10.1109/ISCAS.2010.5537414

[DDD04] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Appl. Math.*, 57:1413–1457, 2004.

[DEN11] Dennis, B. H., Jin, W., Dulikravich, G. S., Jaric, J. Application of the Finite Element Method to Inverse Problems in Solid Mechanics. International Journal of Structural Changes in Solids, Volume 3, Issue 2, 2011, pp. 11-22

[EHN96] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of inverse problems*. Kluwer Acad. Publ., Dordrecht, Netherlands, 1996.

[EKN89] H. W. Engl, K. Kunisch, and A. Neubauer. Convergence rates for Tikhonov regularisation of non-linear ill-posed problems. *Inverse Problems*, 5:523–540, 1989.

[FAR13] C. R. Farrar, K. Worden, *Structural Health Monitoring—A Machine Laerning Perspective*, Wiley, 2013

[FIG10] Figueiredo, E., Park, G., Farrar, C. R., Worden, K., & Figueiras, J. A. (2010). *Machine learning algorithms for damage detection under operational and environmental variability*. Structural Health Monitoring, 10(6), 559–572. doi:10.1177/1475921710388971

[FRA08] Franke, R., Hoffmann, F., & Bertram, T. (2008). *Observation of link deformations of a robotic manipulator with fiber Bragg grating sensors*. 2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 90–95. doi:10.1109/AIM.2008.4601640

[FRI01] Friswell, M. I., Mottershead, J. E. Inverse Methods in Structural Health Monitoring. Key Engineering Materials, Volume 204-205, 2001, pp. 201-210

[GAM11] J. Gama and P. Kosina, "Learning Decision Rules from Data Streams," in Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[GHE10] Ghezzo, F., Starr, A. F., & Smith, D. R. (2010). *Integration of Networks of Sensors and Electronics for Structural Health Monitoring of Composite Materials*. Advances in Civil Engineering, 2010, 1–13. doi:10.1155/2010/598458

[GHE11] Gherlone, M., Cerracchio, P., Mattone, M., Di Sciuva, M., Tessler, A. *Dynamic Shape Reconstruction of three-dimensional frame structures using the inverse Finite Element Method*. Proceedings of the COMPDYN 2011 Conference, Corfu, Greece, May 25th-28th, 2011

[GHW79] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21:215–223, 1979.

[GOU13] P. L. Gould. *Introduction to Linear Elasticity*. Springer New York, 2013.

[GRE08] Gregg, D., Casey, K., Ertl, M. A., & Shi, Y. (2008). Virtual machine showdown. ACM Transactions on Architecture and Code Optimization. doi:10.1145/1328195.1328197

[GUE06] R. Guerraoui, L. Rodigues, *Introduction to Reliabe Distributed Programming*, Springer, 2006

[HAN06] Hanna, D. M., & Haskell, R. E. (2006). Flowpaths: Compiling stack-based IR to hardware. Microprocessors and Microsystems, 30(3), 125–136. doi:10.1016/j.micpro.2005.07.001

[HAN92] P. C. Hansen. Analysis of discrete ill-posed problems by means of the l-curve. *SIAM Review*, 34(4):561–580, 1992.

[HAN95] M. Hanke. *Conjugate gradient type methods for ill-posed problems*. Pitman Reserch Notes in Mathematics. Pitman, 1995.

[HAN97] M. Hanke. A regularizing Levenberg-Marquardt scheme, with applications to inverse groundwater filtration problems. *Inverse Problems*, 13:79–95, 1997.

[HNS95] M. Hanke, A. Neubauer, and O. Scherzer. A convergence analysis of the Landweber iteration for nonlinear ill-posed problems. *Numerische Mathematik*, 72(1):21–37, 1995.

[HED04] Hedley, M., Hoschke, N., Johnson, M., Lewis, C., Murdoch, A., Price, D., & Prokopenko, M. (2004). *Sensor Network for Structural Health Monitoring*. Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. (pp. 361–366). Ieee. doi:10.1109/ISSNIP.2004.1417489

[HPR09] U. Hämarik, R. Palm, and T. Raus. On minimization strategies for choice of the regularization parameter in ill-posed problems. *Numerical Functional Analysis and Optimization*, 30:924–950, 2009.

[HPR11] U. Hämarik, R. Palm, and T. Raus. Comparison of parameter choices in regularization algorithms in case of different information about noise level. *Calcolo*, 48(1):47–59, 2011.

[HR96] M. Hanke and T. Raus. A general heuristic for choosing the regularization parameter in ill-posed problems. *SIAM J. Sci. Comput.*, 17:956–972, 1996.

[HU13] F. Hu and Q. Hao (Eds.), *Intelligent Sensor Networks*. CRC Press, 2013, ISBN 9781420062212.

[HUN01] S. R. Hunt, I. G. Hebden, Validation of the Eurofighter Typhoon structural health

and usage monitoring system, Smart Materials and Structures, Volume 10, 2001, pp. 497.

[JOH97] Johns, David ; Martin, Kenneth W.: Analog Integrated Circuit Design. John Wiley & Sons, Inc., 1997

[Kel95] C. T. Kelley. Iterative Methods for Linear and Nonlinear Equations. SIAM, Philadelphia, 1995.

[KES05] Kester, W.:The Data Conversion Handbook, Analog Devices, 2005

[KIR96] A. Kirsch. An Introduction to the Mathematical Theory of Inverse Problems. Springer, 1996.

[KNS08] B. Kaltenbacher, A. Neubauer, and O. Scherzer. *Iterative Regularization Methods for Nonlinear Ill-Posed Problems*. de Gruyter, Berlin, 2008.

[KOR07] I. Koren, C. M. Krishna, *Fault tolerant Systems*, Morgen Kaufmann, 2007

[LAN51] L. Landweber. An iteration formula for Fredholm integral equations of the first kind. *Amer. J. Math.*, 73:615–624, 1951.

[LAS06] P. Laskov, C. Gehl, and S. Krüger, "Incremental Support Vector Learning: Analysis, Implementation and Applications," Journal of Machine Learning Research, vol. 7, 2006.

[LEH13B] D. Lehmhus, J. Brugger, P. Muralt, S. Pané, O. Ergeneman, M.-A. Dubois, N. Gupta, M. Busse, When nothing is constant but change: Adaptive and sensorial materials and their impact on product design, Journal of Intelligent Material Systems and Structures, Volume 24, Issue 18, pp. 2172-2182

[LEV44] K. Levenberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math*, 2:164–168, 1944.

[LR09] A. Lechleiter and A. Rieder. Towards a general convergence theory for inexact Newton regularizations. *Numer. Math.*, 114:521–548, 2009.

[LUO06] Luo, X., Dong, M., & Huang, Y. (2006). *On distributed fault-tolerant detection in wireless sensor networks*. IEEE Transactions on Computers, 55(1). doi:10.1109/TC.2006.13

[MAR09] I. L. Marques, J. Ronan, and N. S. Rosa, "TinyReef: a Register-Based Virtual Machine for Wireless Sensor Networks," in IEEE SENSORS 2009 Conference, 2009, pp. 1423–1426.

[MAR63] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.

[MAV97] Mavroidis, C., Dubowsky, S., & Thomas, K. (1997). Optimal sensor location in motion control of flexibly supported long reach manipulators. Transactions of the ASME, Journal of Dynamic Systems, Measurement and Control, 119

[MIT12] H. B. Mitchell, *Data Fusion, Concepts and Ideas*, Springer, 2012

[MOR12] G. D. F. Morales, "Big Data and the Web: Algorithms for Data Intensive Scalable Computing," IMT Institute for Advanced Studies, Lucca,, 2012.

[MOR68] V. A. Morozov. The error principle in the solution of operational equations by the regularization method. *USSR Computational Mathematics and Mathematical Physics*, 8:63–87, 1968.

[MOR84] V. A. Morozov. *Methods for Solving Incorrectly Posed Problems*. Springer, New York, 1984.

[MS12] J. Mueller and S. Siltanen. *Linear and Nonlinear Inverse Problems with Practical Applications*. Computational Science and Engineering, vol. 10. SIAM, Philadelphia, 2012.

[MUE07] Müller, R., Alonso, G., & Kossmann, D. (2007). A virtual machine for sensor networks. Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (pp. 145–158). ACM. doi:http://doi.acm.org/10.1145/1272996.1273013

[MUM06] B. Murmann, "LIMITS ON ADC POWER DISSIPATION," in Analog Circuit Design, 2006, pp. 351–367.

[MUR07] B. Murmann, *ADC Performance Survey* 1997–2007 [Online]. Available: http://www.stanford.edu/murmann/adcsurvey.html

[PAN11A] F. Pantke, S.Bosse, D. Lehmhus, and M. Lawo, *An Artificial Intelligence Approach Towards Sensorial Materials*, Future Computing Conference, 2011

[PAN11B] F. Pantke, S.Bosse, D. Lehmhus, and M. Lawo, M. Busse, *Combining Simulation and Machine-Learning for Real-Time Load Identification in Sensorial Materials*, 2nd International Conference on Simulations in Bio-Science and Multiphysics (SimBio-M 2011), Marseille/France, 20-22.6.2011

[PAR11] Park, J., Park, J., Song, W., Yoon, S., Burgstaller, B., & Scholz, B. (2011). Tree-graph-based Instruction Scheduling for Stack-based Virtual Machines. Electronic Notes in Theoretical Computer Science, 279(1), 33–45. doi:10.1016/j.entcs.2011.11.004

[PEE09] B. Peeters, G. Couvreur, O. Razinkov, C. Kündig, H. Van der Auweraer G. De Roeck, Continuous monitoring of the Øresund Bridge: system and data analysis. Structure and Infrastructure Engineering: Maintenance, Management, Life-Cycle Design and Performance, Volume 5, Issue 5, 2009, pp. 395-405

[PLA05] Plassche, Rudy J. d.: CMOS Integrated Analog-to-Digital and Digital-to-Analog-Converters. 2nd Edition. Springer-Verlag New York Inc., 2005

[RAJ13] K. Rajan, Informatics for Materials Science and Engineering, Elsevier, 2013.

[RAL01] L. Ralaivola and F. D'Alché-Buc, "Incremental Support Vector Machine Learning: a Local Approach," in Artificial Neural Networks — ICANN 2001, Georg Dorffner, H. Bischof, and K. Hornik, Eds. Springer, 2001, pp. 322–330.

[REN01] W. J. Renton, Aerospace and structures: Where are we headed? International Journal of Solids and Structures, Volume 39, Issue 17, 2001, pp. 3309-3319

[RIE03] Andreas Rieder. *Keine Probleme mit Inversen Problemen*. Vieweg, 1. edition, 2003.

[ROS07] Roscher, K.-U., Fischer, W.-J., Landgraf, J., Pfeifer, G., & Starke, E. (2007). *Sensor Networks for Integration into Textile-Reinforced Composites*. TRANSDUCERS 2007 - 2007 International Solid-State Sensors, Actuators and Microsystems Conference, 1589–1592. doi:10.1109/SENSOR.2007.4300451

[RUD91] W. Rudin. *Functional Analysis*. McGraw-Hill, 2. edition, 1991.

[RUL13] R. P. Rulli, F. Dotta, P. A. da Silva, Flight Tests Performed by EMBRAER with SHM Systems, Key Engineering Materials, Volume 558, 2013, pp. 305-313.

[RYT93] T. Rytter: Vibration based inspection of civil engineering structure, PhD dissertation, Department of building technology and structure engineering, Aalborg University, Denmark, 1993.

[SAM11] C. Sammut, G. I. Webb (Eds.), Encyclopdia of Machine Learning, Springer, 2011

[SAN13] L. G. dos Santos, EMBRAER Perspective on the Challenges for the Introduction of Scheduled SHM (S-SHM) Applications into Commercial Aviation Maintenance Programs, Key Engineering Materials, Volume 558, 2013, pp. 323-330.

[SCO03] M. D. Scott, B. E. Boser, and K. S. J. Pister, "An ultralow-energy ADC for smart

dust," IEEE Journal of Solid-State Circuits, 2003.

[SKHK12] T. Schuster, B. Kaltenbacher, B. Hofmann, and K. S. Kazimierski. *Regularization Methods in Banach Spaces*. de Gruyter, Berlin, 2012.

[SMI05] Smith, J. E., & Nair, R. N. R. (2005). The architecture of virtual machines. Computer, 38(5). doi:10.1109/MC.2005.173

[SUN09] T. Sundström, B. Murmann, and C. Svensson, "Power Dissipation Bounds for High-Speed Nyquist Analog-to-Digital Converters," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, vol. 56, no. 3, pp. 509–518, 2009.

[SUN10] M. Sun, W. J. Staszewski, R. N. Swamy, Smart Sensing Technologies for Structural Health Monitoring of Civil Engineering Structures. Advances in Civil Engineering, Volume 2010, Article ID 724962, 13 pages, doi:10.1155/2010/724962

[SV89] T. I. Seidman and C. R. Vogel. Well posedness and convergence of some regularization methods for non-linear ill posed problems. *Inverse Problems*, 5:227–238, 1989.

[SYD05] Sydenham, Peter H.; Thorn, Richard: Handbook of Measuring System Design, John Wiley & Sons, Inc., 2005

[TOK03] M. O. Tokhi, M. A. Hossain, M. H. Shaheed, Parallel Computing for Real-time Signal Processing and Control, Springer, 2003

[TRA12] K. Tracht, S. Hogreve, S. Bosse, Intelligent Interpretation of Multiaxial Gripper Force Sensors, Proceedings of CIRP Conference on Assembly Technologies, CATS 2012, 2012

[TRI12] Trigoni, N., & Krishnamachari, B. (2012). *Sensor network algorithms and applications*. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, 370(1958), 5–10. doi:10.1098/rsta.2011.0382

[VAU11] B. Vaugon, P. Wang, and E. Chailloux, "Les microcontrôleurs PIC programmés en Objective Caml," in JFLA, Vingt-deuxièmes Journées Francophones des Langages Applicatifs, 2011, pp. 177–208.

[VAZ06] Vazquez, S. L., Tessler, A., Quach, C. C., Cooper, E. G., Parks, J., Spangler, J. L. *Structural Health Monitoring using high-density Fiber Optic Strain Sensor and Inverse Finite Element Methods*, 2006

[VID11] Vidal-Verdú, F., Barquero, M. J., Castellanos-Ramos, J., Navas-González, R., Sánchez, J. A., Serón, J., & García-Cerezo, A. (2011). A large area tactile sensor patch based on commercial force sensors. Sensors (Basel, Switzerland), 11(5), 5489–507. doi:10.3390/s110505489

[WAH90] G. Wahba. Spline Models for Observational Data. SIAM, 1990.

[WEB14] Webster, John G.; Eren, Halit: Measurement, Instrumentation, and Sensors Handbook, Second Edition, CRC Press, 2014

[WOR07] K. WORDEN and G. MANSON, "The application of machine learning to structural health monitoring," Phil. Trans. R. Soc. A, vol. 365, pp. 515–537, 2007.

[WU99] J. Wu, *Distributed System Design*, CRC Press, 1999

[WUE16] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," PRODUCTION & MANUFACTURING RESEARCH, vol. 4, no. 1, pp. 23-45, 2016.

[YIN13] S. J. C. Ying Z. Lin, Chun Cheng Liu, Guan Ying Huang, Ya Ting Shyu, "A 9-bit 150-MS/s 1.53-mW subranged SAR ADC in 90-nm CMOS," in IEEE Symposium on VLSI Circuits, Digest of Technical Papers, 2013.

[YIP11] M. Yip and A. P. Chandrakasan, “A resolution-reconfigurable 5-to-10b 0.4-to-1V power scalable SAR ADC,” in Digest of Technical Papers - IEEE International Solid-State Circuits Conference, 2011.

[ZH90] H. Y. Zha and P. C. Hansen. Regularization and the general Gauss-Markov linear model. *Math. Comp.*, 55:613–613, 1990.

[ZHA08] X. Zhao, S. Yuan, Z. Yu, W. Ye, J. Cao. (2008), *Designing strategy for multi-agent system based large structural health monitoring*, Expert Systems with Applications, 34(2), 1154–1168. doi:10.1016/j.eswa.2006.12.022

ISBN 978-3-527-33606-7