Intelligent Embedded Systems Laboratory (VIMIMA21)

Laboratory guide for measurement

Distributed Data Acquisition Systems

Written by:
Dr. György Orosz
BME Department of Measurement and Information Systems

July 2023.

Contents

List of Figures

1 Laboratory guide

Distributed Data Acquisition Systems

1.1 Aim of the measurement

Nowadays, we can find more and more different wireless systems in several fielsd. These systems provide a high degree of flexibility due to the wireless transmission of data and usually perform various information gathering and processing tasks. As the units of wireless networks (sensor nodes) are autonomous, the system can be considered as a distributed system. One of the great advantages of distributed systems is that each unit usually has an independent control unit, which can even make local decisions, thus making the system more efficient.

However, wireless communication and distributed signal sensing and processing raise several fundamental problems. The aim of this measurement is to illustrate for the students the typical signal processing problems in distributed wireless signal processing systems. The measurement will provide insight into various signal processing problems by means of a simple system based on wireless sensors. The wireless network consists of so-called MITmote-s using a communication IC of type CC2420 operating in the 2.4 GHz band for radio data transmission.

The measurement is divided in two main parts:

1. Synchronization
2. Resonator-based signal compression (compressive sensing)

The sensor network is connected to a PC, which is used to archive and process the data collected by the sensors.

The problem of synchronisation is in focus throughout the measurement. In the first measurement task, we can learn about a synchronisation technique using a so-called time-stamp transformation for a simple data acquisition system.

The second main task is to learn about a system using data compression with a resonator-based observer. In this case, a wireless sensor generates Fourier-coefficients of the observed signal and transmits only these coefficients on the radio channel. Since these coefficients usually change more slowly than the signal itself, they need to be transmitted less frequently, thus implementing data compression. In addition to synchronisation, this system must also implement the roconstruction of the signal from the measured parameters.

During the measurement, students will gain practice in implementing signal processing tasks, interpreting distributed observations and correcting errors that occur.

1.2 Synchronization

1.2.1 Implementation of time-stamp transformation

Synchronisation is essential in distributed wireless systems. In these cases, the individual sub-units are very loosely connected to each other over the wireless channel, and can therefore be considered as autonomous units, not under *direct* central control.

A typical and fundamental task in wireless systems is the monitoring of certain analogue signals (brightness, temperature, sound, etc.). However, the evaluation of the measurements also requires appropriate time information associated with the observations. For example, when measuring the speed of an object, it is not sufficient to know its position, but it is essential to know at what times the object is at a given location. Since in distributed systems each unit usually has its own clock, providing time information is not a trivial task, especially for rapidly changing signals.

The time measurement is usually based on a quartz oscillator, which is a good compromise between price, accuracy and stability. (Such quartz oscillators are found in many watches, for example.) To get an insight to the orders of magnitude of error, it is important to know that the maximum expected error of a general purpose quartz is in the order of tens of tens of 10 ppm (ppm=parts per million=10^{-6}). This means that, for example, a 10 ppm error for a 10 MHz oscillator will deviate from the nominal value by a maximum of ± 100 Hz. Even this relatively small error can cause a significant deviation in time measurement over a long period of time. For example, if we assume that the clocks of two sensors are the same at the beginning of the day, but the difference in the frequencies of the quartz clocks is 10 ppm, then by the end of the day the difference in the time shown by their clocks will be $24 \cdot 60 \cdot 60 \cdot 10 \cdot 10^{-6} = 0.864$sec, so there is a difference of almost one second. Such an error is unacceptable in a precise measurements.

One obvious way to synchronise the sensors is to tune the clocks all the time (for example, when the noon bell rings, we set our watches to exactly noon). However, tuning the clocks is not always possible. In this case, for example, a synchronisation algorithm implementing time stamp transformation can be used. A timestamp is the time data that is assigned to an event, e.g. when an observation occurred.

Before learning about the algorithm, some definitions need to be clarified. In the following t denotes the reference time, which is the absolute exact time (e.g. the time provided by an atomic clock). $h_i(t)$ denotes the time provided by the clock of the i-th node. $O_{i,j}(t) = h_i(t) - h_j(t)$ is the offset between the clock of the i-th and j-th node, i.e. the time difference. $\rho_{i,j}$ is the drift between the clocks of the i-th and j-th nodes, which gives the rate of change of time difference between the two clocks: $\rho_{i,j} = \frac{dO_{i,j}}{dt} \approx (O_{i,j}(t_2) - O_{i,j}(t_1))/(t_2 - t_1)$. The latter estimate is true if the frequency error of the quartz oscillators is constant. This is usually true (at least in the short term), in which case the drift is equal to the error of the quartz. We can also define the speed of the clocks: $f_i = \frac{dh_i}{dt}$, which shows how fast time passes at the i-th node compared to the reference time. Ideally, $f_i = 1$, so the clock is running at unit speed. The speed relative to each other can be defined in a similar way: $f_{i,j} = \frac{dh_i}{dh_j}$.

Example 1: The times shown by two clocks (node_1 and node_2) are recorded at the following times:

At [Jan 10, 2000 12:00:00].

- the clock of node_1 shows [Jan 10, 2000 12:00:10]
- the clock of node_2 shows [Jan 10, 2000 12:00:09].

At [Jan 15th 2000 12:00:00].

- the clock of node_1 shows [Jan 15th 2000 12:00:11]
- the clock of node_2 shows [Jan 15th 2000 12:00:08].

In this example, the reference time in the first case is: t_1 =[Jan 10, 2000 12:00:00], so:

t_1 =[Jan 10, 2000 12:00:00]:

- the local time of the first node: $h_1(t_1)$ =[Jan 10, 2000 12:00:10]
- the local time of the second node: $h_2(t_1)$ =[Jan 10, 2000 12:00:9]
- the offset between the two clocks is: $O_{1,2}(t_1) = h_1(t_1) - h_2(t_1) = 1\text{sec}$

The reference time in the second case is t_2 =[Jan 15, 2000 12:00:00], so:

t_2 =[15 Jan 2000 12:00:00]:

- the local time of the first node is: $h_1(t_2)$ =[Jan 15th 2000 12:00:11]
- the local time of the second node is: $h_2(t_2)$ =[Jan 15th 2000 12:00:8]
- the offset between the two clocks is therefore: $O_{1,2}(t_2) = h_1(t_2) - h_2(t_2) = 3\text{sec}$.

The time difference between the two times is: $\Delta t = t_2 - t_1 = 5$ day $= 432000$ sec. The difference between the clocks of the two nodes has therefore changed by $(3-1)$ sec during time Δt, so the drift of the two clocks (how fast they move away from each other): $\rho_{i,j} = (3-1)\text{ sec}/\Delta t = 4.63 \cdot 10^{-6} = 4.63$ ppm.

Note that the clocks don't show time and date in the traditional sense in every cases. This time information can be, for example, the time elapsed from the time instant they have been switched on, as we will see in the case of the system used in the measurement.

Continuing the example, suppose that the two clocks are placed at two ends of a bridge and they measure the time it takes for an object to drive over the bridge. The 1. clock reported that a car drove onto the bridge at $h_1(t_3)$=[25 Jan 2000 12:00:13] and the 2. clock reported that the car drove off at $h_2(t_4)$=[25 Jan 2000 12:00:11]. How long did it take the car to drive over the bridge?

To solve this, we need to convert the two clocks to a common time scale, so we need to perform a timestamp transformation. From the previous calculations, we know that at [15 Jan 2000 12:00:00] the offset between the clocks is $O_{1,2} = 3\text{sec}$ and we can see that the offset changes by 2sec over 5 days, so at [25 Jan 2000 12:00:13] the offset is about $3\text{sec} + 2 * 2\text{sec} = 7\text{sec}$. Therefore, if we take the time scale of the first clock as a time basis, we can conclude that the car drove to the bridge at $h_1(t_4) = h_2(t_4) + 7$ sec=[25 Jan 2000. 12:00:18] according to the first clock, which means that the car drove over the bridge within $h_1(t_4) - h_1(t_3) = 5$ sec. (Note that we have made some reasonable simplifications here, e.g., we do not take into account that the two clocks drift slightly relative to each other even during the time it takes the car to cross the bridge, but this does not cause a noticeable error in this case.) We can see that since the clocks are not ideal, they run independently of each other, so it is not possible to use their time data (timestamps) directly, but a times-tamp transformation, is required. In the following section, the mathematical basis of this transformation is presented, which can be used to perform the transformation in a simple way.

In general, for two units, the local times of each unit can be written as a function of each other:

$$h(t)_i = g(h_j(t)). \tag{1}$$

A practical relation is obtained if we assume that local times are linear functions of each other, in which case:

$$h_i(t) = ah_j(t) + b, \tag{2}$$

The linear function means that we take into account that the frequency error of the quartz oscillators of the clocks is constant. This is generally true, at least short term. If we assume that (2) is true, and we know the parameters a and b, then we can make the conversion between the clocks of the i-th and j-th nodes explicitly. This operation is called a timestamp transformation.

In this way, we can obtain consistent time results, and we can interpret events detected by two separate sensors. Of course, the method can also be applied to multiple units, in which case it is necessary to choose what the common time scale should be.

The parameters (2) can be defined using so-called synchronization points, which are pairs of time stamps belonging to the same time instant. In our example, these timestamp pairs are $\{h_1(t_1), h_2(t_1)\}$ and $\{h_1(t_2), h_2(t_2)\}$. Based on the two synchronization points, two parameters of the equation (2) can be determined: a and b. In general, several synchronization points are used to determine the parameters, so that the effect of measurement errors is reduced. For a first-order equation, for example (2), linear regression can be used to determine the parameters. An example of this is shown in 1. MATLAB provides efficient help for curve fitting to various degrees using the `polyfit` command.

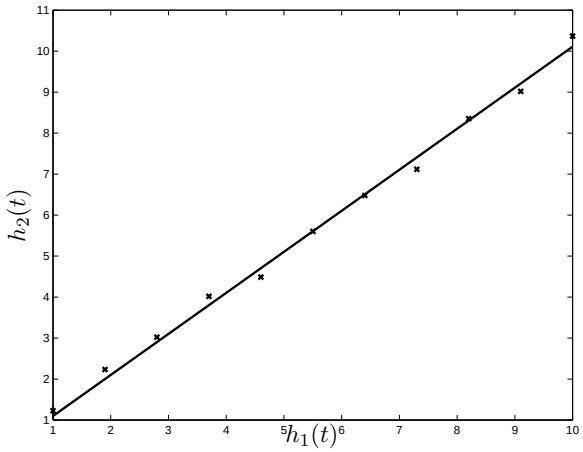


Figure 1: Local time functions and synchronisation points. Continuous line: $h_2(t) = ah_1(t) + b$, $\times$: $\{h_1(t_k), h_2(t_k)\}$ synchronization poit pairs.

In general, the timestamp transformation can be performed in three steps:

1. Collecting synchronisation points.
2. The synchronisation points are used to determine the parameters needed to convert the timestamps: a and b in equation (2).
3. Time stamp conversion based on the parameters defined in the previous section and equation (2).

1.2.2 Synchronisation with interpolation

Besides the non-synchronized time bases of the individual units, another problem of distributed signal detection is that the data acquisition, i.e. the sampling of the signals, is done more or less independently of each other on each device. In many cases, however, not only data with correct time stamps but also data that are coherent in time are needed (i.e., sampled data should belong to the same time instant).

Example 2: Two sensors (node_1 and node_2) measure the pressure of two tanks ($p_1(t)$ and $p_2(t)$) independently. In a given minute, the following measurement results are obtained for the first tank: $\{17\text{ sec}, 5\text{ bar}\}$, $\{38\text{ sec}, 5.6\text{ bar}\}$, $\{59\text{ sec}, 5.8\text{ bar}\}$. For the other tank: $\{10\text{ sec}, 3\text{ bar}\}$, $\{33\text{ sec}, 3.8\text{ bar}\}$, $\{50\text{ sec}, 4.0\text{ bar}\}$. The graphs are shown in Figure 2. What is the pressure difference between the tanks at a given minute $t_p = 30$sec. (Assume that the time scale is already correct.)

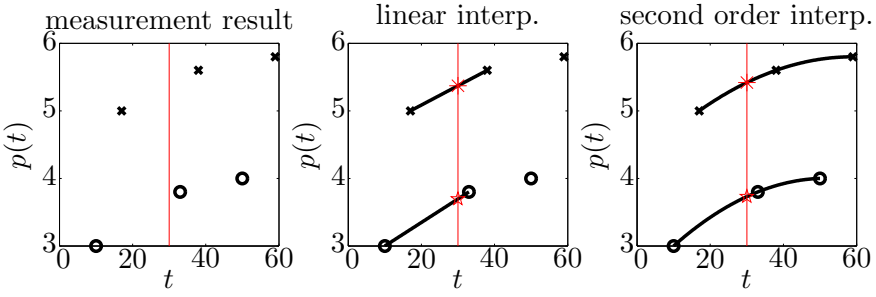


Figure 2: Measured pressure values, and interpolation with linear and second order interpolation. $\times$: $p_1(t)$, $\circ$: $p_2(t)$

It can be seen that there is no measurement result at $t_p = 30$ sec, so the pressure values have to be estimated somehow. An obvious method is to fit some polynomial to the measurement points. An example is shown in Figure 2. The simplest method is linear interpolation. In this case the measured signal is estimated by a line between the measurement results. The mathematical description of this is as follows. Given the value of a quantity at time T_1 and T_2: $x(T_1)$ and $x(T_2)$. Let the difference between T_1 and T_2 be $T_p = (T_2 - T_1)$. We can estimate the quantity x at time $t_p \in [T_1, T_2]$ as follows:

$$x(t_p) = x(T_1)\frac{T_2 - t_p}{T_p} + x(T_2)\frac{t_p - T_1}{T_p}. \tag{3}$$

Thus, each measurement result is weighted in proportion to how far the time t_p in question is from the sampling time.

In our example, the method gives the following result: $p_1(30) = 5\text{ bar} * \frac{38-30}{38-17} + 5.6\text{ bar} * \frac{30-17}{38-17} = 5.37$ bar and $p_2(30) = 3\text{ bar} * \frac{33-30}{33-10} + 3.8\text{ bar} * \frac{30-10}{33-10} = 3.7$ bar.

Higher degree polynomials can of course be used for the estimation. In this case, a polynomial is fitted to the points in the vicinity of the time in question and evaluated at the location in question. For fitting higher degree polynomials, MATLAB functions `polyfit` and `polyval` are useful.

Data points `x` and `y` should be given for the function `polyfit`. `x` and `y` are data points on which we want to fit a polynomial of degree `n`, and the function returns the coefficients of the fitted polynomial in a vector `coef`: `coef = polyfit(x,y,n)`.

Using second-order interpolation in the previous example, we can use the following MATLAB code to fit a second-order polynomial:

```
fitPoly_1 = polyfit([17 38 59],[5 5.6 5.8],2);
            %fitting polinomial on data points
P_1 = polyval(fitPoly_1,30) %fitted polynomial
          %evaluation at a given point
P_1 =
```

```
      5.4186

fitPoly_2 = polyfit([10 33 50],[3 3.8 4.0],2);
P_2 = polyval(fitPoly_2,30)
P_2 =
      3.7302
```

Note that to fit a polynomial of order n, at least $n+1$ points must be given for function `polyfit`. During the measurement, the difference of the time values may be relatively small compared to their values. For example, if the time values in the second case are: [10000000010 10000000033 10000000050] and the time in question is 10000000030, the calculation would be very inaccurate due to numerical reasons; MATLAB warns the user for this. In this case, two solutions are suggested: shift the values, for example by subtracting the first element from each of them, so that the difference is relatively large, or rescale (multiply by a constant) the values to a reasonable order of magnitude. This does not change the value after fitting, since the whole function is simply shifted along the x axis or stretched proportionally. Then we get the following code:

```
t=[10000000010 10000000033 10000000050] %time instant of measurement
tp=10000000030 %time instant where function should be estimated
x=[3 3.8 4.0]
scaleF=1000; %scale factor
          %also both for fitting and scaling
fitPoly_2 = polyfit( (t - t(1))*scaleF, x, 2 );
P_2 = polyval( fitPoly_2, (tp-t(1))*scaleF )
P_2 =
      3.7302
```

In the code above, for the sake of simplicity and generality, the times are placed in `t` and `tp`, and the offset is done with `t(1)`, which is the first element of the vector `t`. The scale factor `scaleF` was used. It can be seen that the final result is not affected by either the offset or the scaling. If MATLAB considers these solutions (scaling, shifting) to be poorly conditioned, the warning messages can be turned off using the following command:

`warning('OFF', 'MATLAB:polyfit:RepeatedPointsOrRescale');`. Attention! This does not eliminate the problem, it turns off only the error messages, which, if displayed frequently, will slow down the program considerably. You can use this option, if no other method has solved the problem!

The interpolation algorithm is implemented as follows. See the timing diagram in Figure 3. Suppose we want to calculate the value measured by the sensor at time t_p. The value measured by the sensor at time t is denoted by $s(t)$. Suppose that we want to perform the interpolation with a polynomial of order n, then we need at least $n+1$ of data. For simplicity, let n be odd.

1. Let's find the nearest sampling point of sensor close to the gateway's sampling time t_p. This time instant is: t_r.

2. Let's create a vector of size $n+1$ from the timestamps and measurement results of the sensor. These vectors are: $T = [t_{r-k}, t_{r-k}...t_{r+k}]$ and $S = [s(t_{r-k}), s(t_{r-k})...s(t_{r+k})]$, where $n \leq 2 \cdot k$.

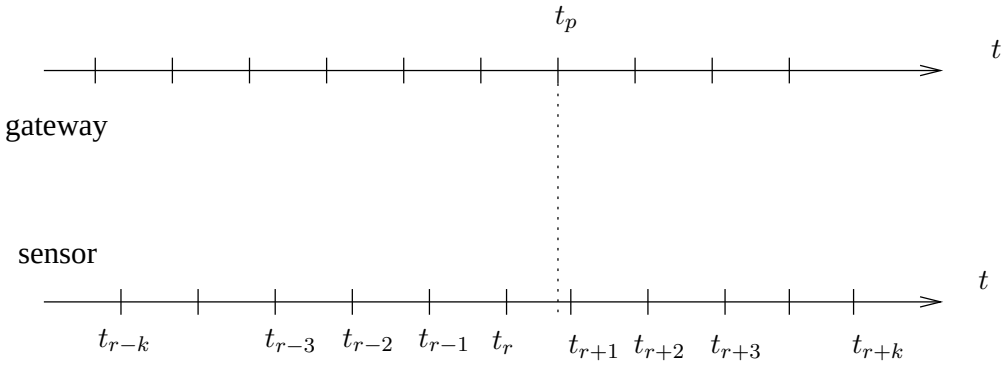


Figure 3: Timing diagram of the algorithm

3. Let's fit a polynomial on data points T and S. (Scaling and shifting can be used, e.g., t_r can be subtracted from T and can be multiplied with a constant term. The same operation should be performed for t_p!)

4. Let's evaluate the polinomial at time instant t_p.

1.2.3 The Measurement System

In the first part, we use the measurement arrangement shown in Figure 4.

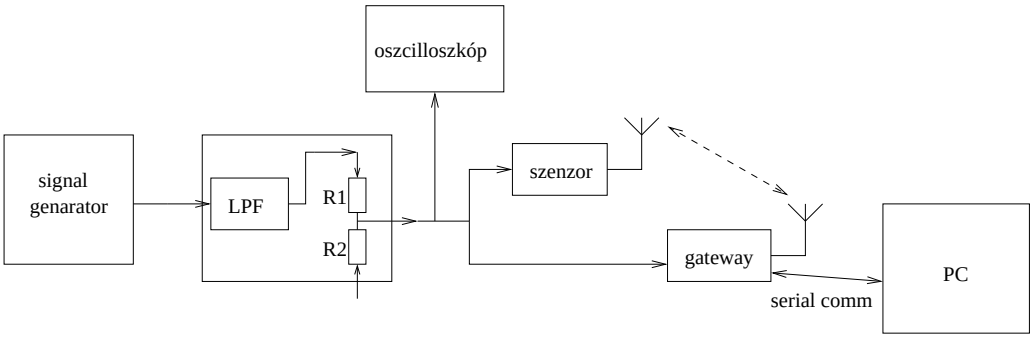


Figure 4: Block diagram of the measurement system (both for a sensor with sampling and a sensor with resonator-based observer)

In this system there are two MITMOT-s (see Figure 9.) One node is a sensor node (hereafter called sensor). This node samples the signal with the AD converter and transmits it via radio to the node that acts as a gateway. The gateway node receives the data and transmits it to the PC via a serial port. The gateway node also serves as a sensor and, like the sensor, samples the signal with its AD converter and sends also its own data to the PC via a serial port. In the first part of the measurement, the same signal is fed to the inputs of the sensor and the gateway to check the accuracy of the synchronisation: both nodes should detect the same signal, apart from differences due to the analogue electronics. The signal from a function generator is used as an external signal. This signal is fed to a simple low pass filter (LPF), which acts as a band limiting filter to reduce the effect of overlapping due to the finite sampling frequency (see later for the role of resistors R1 and R2, they have no effect in this measurement.) The filter is shown in Figure 10.

You will see later that the sensor can operate in two modes (sampling or preprocessing). After switching on the sensor, the mode is set to mode 1 by pressing SW1 and SW2 (the 7-segment display shows the mode number). Once selected, the mode can be confirmed by a long press on SW3.

The nodes have the same nominal sampling frequency, 1800.2 Hz, more exactly: $f_s = (8 \cdot 10^6/4444)$ Hz.

Data transmission is packet-based. Both sensor and gateway data are transmitted to the PC in 25 sample-size packets. Each packet is provided with a time stamp indicating the sampling time of the first sample in the packet. This time represents the time elapsed since the time instant when the node was switched on.

The gateway node also transmits the synchronisation points to the PC, which can be used to perform the timestamp transformation. The synchronisation points are generated by means of radio communication, taking advantage of the radio IC (CC2420). A special feature of the CC2420 IC is that at the start of transmission, the IC generates a rising edge at a digital output (SFD pin: Start of Frame Delimiter) on the transmitter and receiver side (see datasheet for details). This rising edge is generated with very high precision at the same time on the transmitter and receiver side, so if both the transmitter and the receiver record this time, the resulting timestamp pairs can be used as synchronisation points. The time associated with a given transmission is still stored in the current packet by the transmitter, so that the receiver can transmit it, together with the timestamp it has recorded itself, to the PC, where these synchronisation points can be used to perform the timestamp transformation.

The data sent by the gateway node can be saved to the hard disk using a PC program. The saved files can be converted to a format that can be easily processed by calling the `preprocessSampleFiles.m` file available in the sample.

Figure 5: Graphical user interface of the simple data acquisition system.

After preprocessing, the following files can be used for the measurement.

pre_stmpSync.dat: the file containing the synchronization point pairs. The file consists of two columns, each row contains a synchronization point pair. The first column contains the time stamps belonging to the gateway and the second column contains the time stamps belonging to sensor. One column belongs to the same radio transmission. An example of a possible result:

4068.618299375 3.014941125
4068.632187125 3.028829000
4068.646074750 3.042716625
4068.659961625 3.056603625
4068.673849375 3.070491625
4068.687732125 3.084374500
4068.701622875 3.098265250
4068.715510625 3.112153125
4068.729398375 3.126041125

So the first radio transmission occured 4068.618299375 sec after the gateway has been turned on, and after 3.014941125 sec the sensor has been turned on. These two time instants belong to the same event.

The file (and usually the others) can be easily loaded using the MATLAB load command, for example:

```
readSync = load('pre_stmpSync.dat');
```

will return a vector of two columns matching the contents of the file.

pre_gwySampFull.dat, **pre_sensorDataFull.dat**: contains the data sent by the gateway and the sensor and the corresponding timestamps. The second column contains the samples, and the first column contains the timestamp associated to the sample in the first column.

1.3 The Resonator-based Measurement System

In the second part of the measurement, a measurement system using resonator-based compression is investigated. The block diagram of the measurement system is the same as in Figure 4, i.e. a sensor and a gateway node (the gateway also performs data acquisition). The difference between the two systems is that while in the first case the sensor sampled the signal and transmitted the raw samples, in this measurement the sensor generates the Fourier-coefficients of the signal and transmits the Fourier-coefficients instead of the raw samples. This system can be used for periodic signals, since the Fourier-coefficients can be used to generate periodic signals. The advantage of this system is that for periodic signals, the amount of data to be transmitted over the radio network can be reduced, as the Fourier-coefficients of periodic signals tend to vary more slowly than the signal itself. In our case, the sensor transmits half as much data as a simple sampling sensor.

The system demonstrates the potential offered by smart sensors, i.e. the sensor is capable of performing certain tasks even on the sensor side. This computational power can be used to pre-process the measured signals, which can help to reduce the load on the central unit or, as in our case, the communication load can be reduced, since it is sufficient to transmit only some signal parameters not the raw samples. In the following chapters, we will learn about the architecture of the resonator-based observer, which is one of the basic building blocks of the system.

1.3.1 Introduction to resonator-based observer

This section introduces the Resonator-Based Observer (RBO). Although a detailed knowledge of the algorithm is not necessary for the measurement, we will only use it at the "application" level, a brief overview of the algorithm is given to give an insight into the system.

The resonator-based observer is used to recursively compute Fourier-coefficients of periodic signals. The algorithm is a potential alternative to the traditional frequency domain discrete Fourier transforms (DFT), but the two algorithms have the following fundamental differences:

1. The DFT calculates the spectrum of the signal uniformly along the entire frequency axis, while the RBO is only suitable for calculating the Fourier-coefficients of periodic signals. This represents a significant saving of resources, since the spectrum does not need to be computed at locations where no useful component is found.

2. The RBO requires knowledge of the frequency of the signal, while DFT does not. If the frequency of the signal is known, RBO eliminates the picket fence and leakage phenomena that occur with DFT.

3. The BRO recursively produces Fourier-coefficients, so it updates the estimate from sample to sample. In contrast, the DFT performs block processing, so all the data needed for the transformation must be collected before the transformation is performed, which usually introduces a larger delay for a given accuracy.

Before introducing the RBO algorithm, we will describe how to represent the periodic signals, which is necessary because RBO measures the parameters of periodic signals. Given a periodic signal y_n with P number of harmonics, Θ is the relative frequency with respect to the sampling frequency, and $x_{i,n}$ is the Fourier-coefficients associated with the i-th harmonic at time n. Then

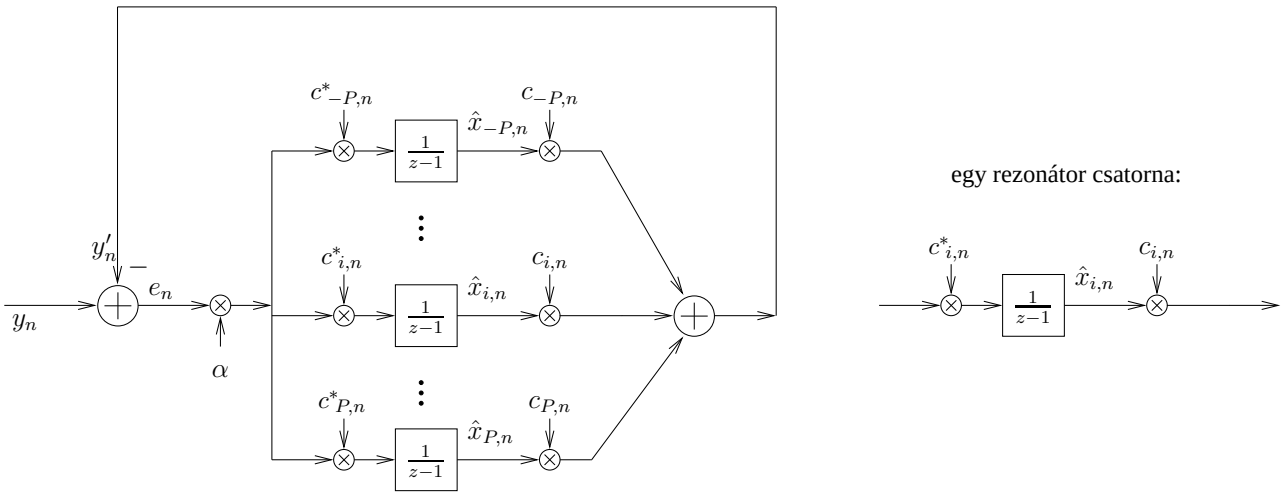


Figure 6: Block diagram of RBO

the signal y_n can be written in the following form:

$$y_n = \sum_{i=-P}^{P} x_{i,n} e^{\jmath \Theta i n} = \sum_{i=-P}^{P} x_{i,n} c_{i,n}, \quad \text{and} \quad c_{i,n} = e^{\jmath \Theta i n} \tag{4}$$

where n is the time index, and $c_{i,n} = e^{\jmath \Theta i n}$ is the basis function for the ith harmonic. The above equation is valid in general sampling systems. If we use a real time scale, the following holds:

$$y(t) = \sum_{i=-P}^{P} x_i(t) e^{\jmath f i t} = \sum_{i=-P}^{P} x_i(t) c_i(t). \tag{5}$$

In this case, f denotes the frequency, thus assuming a sampling frequency of f_s: $\Theta = f/f_s$. The two types of description –(4) and (5)– are necessary because during the measurement we deal with systems with distributed operation, where in general we cannot speak about global sampling times, but each unit has local sampling times. Thus, e.g. (4) is better used to describe the algorithms running on individual units, but the interpretation of the data exchanged between each other is only possible with an absolute, global time scale, as we see in the case of (5).

The equation (4) also has a more compact form, which simplifies the description. In this form, we use the following notations:

$$\begin{aligned} \mathbf{x}_n &= [x_{-P,n} \quad \dots \quad x_{i,n} \quad \dots \quad x_{P,n}]^{\mathrm{T}} \quad : \text{ column vector} \\ \mathbf{c}_n &= [c_{-P,n} \quad \dots \quad c_{i,n} \quad \dots \quad c_{P,n}]^{\mathrm{T}} \quad : \text{ column vector,} \end{aligned} \tag{6}$$

where $\mathbf{c}_n$ is the vector containing the basis functions at the time instant n and $\mathbf{x}_n$ is the vector containing the Fourier-coefficients at the time instant n. With these notations, using the rules of vector multiplication[1], (4) can be written in the following simple form:

$$y_n = \mathbf{c}_n^{\mathrm{T}} \mathbf{x}_n. \tag{7}$$

[1] $\mathbf{a}^{\mathrm{T}}\mathbf{b} = \sum_i a_i b_i$

The RBO estimates the Fourier-coefficients $\mathbf{x}_n$ of the signal y_n. The structure of the algorithm is shown in Figure 6. We denote the vector containing the estimates of the Fourier-coefficients $\mathbf{x}_n$ by $\hat{\mathbf{x}}_n$. The observer is described by the following equations:

$$y'_n = \mathbf{c}_n^{\mathrm{T}} \hat{\mathbf{x}}_n, \tag{8}$$

$$e_n = y_n - y'_n = y_n - \mathbf{c}_n^{\mathrm{T}} \hat{\mathbf{x}}_n, \tag{9}$$

$$\hat{\mathbf{x}}_{n+1} = \hat{\mathbf{x}}_n + \alpha e_n \mathbf{c}_n^*. \tag{10}$$

(Let's note that there are different notations. It depends on the application which notation is more practical, but the notation does not affect the basic properties.) * denotes complex conjugation. α is a gain factor, we will not go into its definition. Qualitatively, it can be said that a small value of α results in good interference suppression against measurement noises, a large value of α ensures fast adjustment; the specific choice is the result of a compromise. It can be seen that the algorithm follows the usual form of recursive estimation algorithms:

1. Based on the available estimated parameters $\hat{\mathbf{x}}_n$, it provides an estimate of the signal y_n. See y' in equation (8).
2. The estimation error is calculated: (9).
3. The estimated parameters $\hat{\mathbf{x}}_n$ are updated using the error e_n: (10).

The algorithm can be interpreted in several ways. In the following points, we list some ways of interpretation, to make it easier to get to know/understand the algorithm:

1. According to the traditional interpretation, we start from considering the signal y_n as the output of a linear system, whose state equations are:

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n \\ y_n &= \mathbf{c}_n^{\mathrm{T}} \mathbf{x}_n. \end{aligned} \tag{11}$$

   We can consider (11) as the state-equation describing a function generator as a physical system. The state variables of this linear system are the Fourier-coefficients of the observed signal, so the state observer (RBO) designed for this system produces the Fourier-coefficients of the signal.

2. The algorithm is formally very similar to the LMS algorithm, and can really be interpreted as a complex-valued LMS algorithm. In this case, $\mathbf{c}_n$ can be understood as a reference signal, and the goal is to approximate the signal y_n as closely as possible with the base functions $\mathbf{c}_n$ in mean square sense, thus minimizing the following cost function: $J(n) = |e_n|^2 = \left|y_n - \mathbf{c}_n^{\mathrm{T}} \hat{\mathbf{x}}_n\right|^2$. The parameter vector $\hat{\mathbf{x}}_n$ is modified in each time instant in the direction of the negative gradient of the cost function using a convergence parameter α. The negative gradient (taking into account that $\hat{\mathbf{x}}_n$ contains complex numbers) is: $-\nabla J(n) = -\frac{\partial J(n)}{\partial \hat{\mathbf{x}}_n} = \mathbf{c}_n^* \left(y_n - \mathbf{c}_n^{\mathrm{T}} \hat{\mathbf{x}}_n\right) = \mathbf{c}_n^* e_n$. So $\hat{\mathbf{x}}_{n+1} = \hat{\mathbf{x}}_n - \alpha \nabla J(n)$ results in equation (10).

3. The individual resonator channels have infinite gain at the given resonator frequency, so they can be interpreted as integrators shifted to resonator frequencies: multiplication by $c_{i,n}^*$ shifts the signal from the ith resonator frequency to DC, where it is integrated, then multiplication by $c_{i,n}$ shifts the integrated signal back to the original frequency. It is known

from control technology that a feedback integrator can follow a constant signal without error, so all periodic signals found at resonator frequencies can be produced without error with the help of infinite gain shifted to resonator frequencies.

In the actual system, the sensor is capable of calculating a maximum of 5 Fourier-coefficients. This value is limited by the computing power of the microcontroller (ATmega128 @ 8 MHz). The node can also be configured to calculate only the odd Fourier-coefficients. This is useful in cases where the signal only contains Fourier-coefficients in odd harmonic positions (e.g.: triangular signal, square signal). In this case, the calculation of even harmonics would require unnecessary computing power. The DC component is not calculated by the node, since the interface card belonging to the node is AC-coupled. The sensor node can be configured with switch 1 on the IO board to count and transmit the first five harmonics (`ON`) or the first five *odd* harmonics (`OFF`).

Since the observed signal y_n is real-valued, the following equality holds:

$$x_{i,n} = x^*_{-i,n}, \tag{12}$$

therefore, the Fourier-coefficient found at negative frequencies is the conjugate of the components found at positive frequencies, therefore the sensor transmits only the components $\{x_{i,n};\ i > 0\}$, since a real and DC-free signal y_n can be clearly generated from these data as well:

$$y_n = 2\Re\left\{\sum_{i=1}^{P} x_{i,n}c_{i,n}\right\}, \tag{13}$$

where $\Re\{\cdot\}$ denotes the real part of a number. (13) can be obtained from equations (4) and (12) using that for a complex[2] number $\gamma : \gamma + \gamma^* = 2\Re\{\gamma\}$

Frequency measurement with RBO The frequency of the signal must be specified for the observer. A PC user interface is used for this during measurement (see later in Figure 8). The exact measurement of the frequency is based on the measurement results of the sensor, using the basic idea of the Adaptive Fourier Analyzer (AFA). In this we apply the following relation. Let a signal with frequency ω_{jel} and phase ϕ_{jel} be given: $y(t) = \cos(\omega_{jel}t + \phi_{jel})$. If we describe a signal $y(t)$ with a basis function of frequency ω_1, then we obtain the following equation: $y(t) = \cos[\omega_{jel}t + \phi_{jel}] = \cos\{\omega_1 t + [(\omega_{jel} - \omega_1)t + \phi_{jel}]\}$, which is:

$$y(t) = \cos\{\omega_1 t + [(\omega_{jel} - \omega_1)t + \phi_{jel}]\} = \cos\left\{\omega_1 t + \phi'_{jel}\right\}, \tag{14}$$

So we get a signal with frequency ω_1 whose phase changes with the rate proportionally to the difference of the two frequencies:

$$\phi'_{jel} = (\omega_{jel} - \omega_1)t + \phi_{jel} = 2\pi(f_{jel} - f_1)t + \phi_{jel}. \tag{15}$$

If the frequency of the resonators is set to an estimated frequency, the deviation of the signal frequency from the given frequency is shown by the fact how fast the phase of the observed signal changes (this means that the Fourier-coefficient of the observed signal rotates with angular frequency proportional to the frequency deviation). If we plot the phase, the slope of the line fitted to the phase gives the frequency deviation (the slope is to be interpreted in $\frac{\text{rad}}{\text{sec}}$, so it must be divided by 2π to get the frequency). The phase of a complex number is possible with

[2] if $\gamma = a + \jmath b$, then $\gamma^* = a - \jmath b$, so: $\gamma + \gamma^* = a + \jmath b + a - \jmath b = 2a$, hence twice of the real part of γ

the MATLAB `angle` command, and it is also useful to use the `unwrap` command (for details, see MATLAB: `help unwrap`).

The frequency can be found manually by observing the phase of the Fourier-coefficient of the fundamental harmonic (shown on the user interface in Figure 8), and if it rotates counter-clockwise, then the frequency must be increased, otherwise it must be decreased. The speed of the rotation is proportional to the frequency error: if the phase changes by $d\phi$ degrees between the arrival of two consecutive Fourier-coefficients, which is $T_F \approx \frac{1}{36}$ sec, then the frequency difference is: $\frac{d\phi}{360^\circ \cdot T_F}$. The value of $d\phi$ can be read directly from the user interface shown in Figure 8. `dFi` in the figure.

1.3.2 Synchronization and signal reconstruction in a system using a resonator observer

During the measurement, our goal is to restore the measured signal from the Fourier-coefficients provided by the sensor and then transmitted to the PC via the gateway node. The reconstructed signal can be compared with the data sampled by the gateway node, so we can test the properties of the algorithm.

It is important to note that the time scale of the sensor node must also be transformed in this system. The synchronization points are available for time-stamp transformation as described in Section 1.2.3.

The sensor node transmits the Fourier-coefficients to the PC every 50 sampling times (T_F). Since a sampling interval is: $T_s = 4444/8 \cdot 10^6 \approx 0.5555$ msec so the frequency of sending the coefficients is: $T_F = 50 \cdot 4444/8 \cdot 10^6$ sec $\approx \frac{1}{36}$ sec ≈ 27.8 msec. The data package containing the Fourier-coefficients contains the following data:

1. Package ID.
2. Transmission time instant of the package denoted by t_r.
3. Five Fourier-coefficients.
4. The value of the basis function at the time instant of the transmission time of the package which is: $\varphi_1(t_r)$.

The latter is necessary because, in the case of a periodic signal, a phase information alone does not say anything, a reference must be provided. If we specify the phase of the fundamental harmonic basis function at a given time, the phase and the basis function can be calculated at any other time if the frequency f is constant:

$$\varphi_1(t) = \varphi_1(t_r) + 2\pi f(t - t_r), \tag{16}$$

Since the frequency can change, it is necessary to send the phase with a certain frequency. The phase of the ith harmonic basis function is i times the phase of the fundamental harmonic:

$$\varphi_i(t) = i\varphi_1(t), \tag{17}$$

The ith basis function is:

$$c_i(t) = e^{j\varphi_i(t)}. \tag{18}$$

During the measurement, the sensor and the gateway node sample the same signal (see Fig. 4). The gateway transmits the raw samples meanwhile the sensor transmits the Fourier-

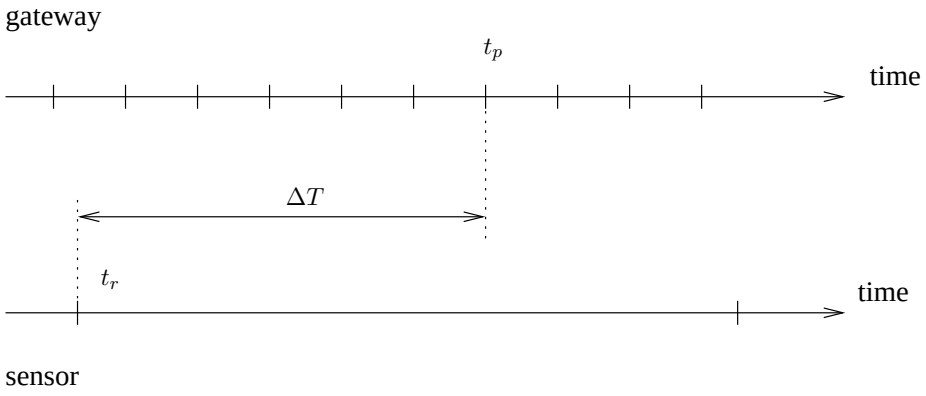


Figure 7: The timing diagram of signal reconstructuon from Fourier-coefficients.

coefficients. Using the Fourier-coefficients, the observed signal can be reconstructed at time instant t_p by performing the following steps (for an explanation of time data, see Fig. 7):

1. Measure and set the frequency of the signal, then perform a measurement.
2. Let's do the sensor timestamp conversion.
3. To generate the signal at point t_p, we search for the last packet arriving immediately before time t_p. Time of sending the packet: t_r.
4. Let's calculate the time since the package arrived: $\Delta T = t_p - t_r$.
5. The value of the fundamental harmonic phase given at the time instant when sending the packet ($\varphi_1(t_r)$) is modified by the change of the phase during the time ΔT, thus obtaining the phase of the fundamental harmonic basis function at time t_p: $\varphi_1(t_p) = \varphi_1(t_r) + 2\pi\Delta T \cdot f$ (see: (16)).
6. Let's calculate the phase of the i-th harmonic basis function: $\varphi_i(t_p) = \varphi_1(t_p) \cdot i$ (see: (17)).
7. Let's calculate the basis functions: $c_i(t_p) = e^{j\varphi_i(t_p)}$ (see: (18)).
8. Knowing the basis functions and the Fourier-coefficients (x_i) sent by the sensor, calculate the value of the signal based on (13): $y'(t_p) = 2\Re\left\{\sum_{i=1}^{P} c_i(t_p)x_i\right\}$.

Attention! In MATLAB, the $\sum$ operation can be efficiently implemented by vector multiplication, see, e.g., equation (7), in which the transpose of one vector ($^{\mathrm{T}}$) must be calculated. A common error is that in MATLAB the [.'] operator denotes transposition, the ['] operator denotes the transposed conjugate, but this does not cause problems for real signals, only for complex signals. Also note that if `i` or `j` variables are used as cycle variables, they cannot be used to denote the imaginary unit. Instead, `1i` or `1j` is recommended to denote the imaginary unit.

1.3.3 System description

A graphical user interface, see Figure 8, is used to manage the system using an RBO. The user interface offers the following options:

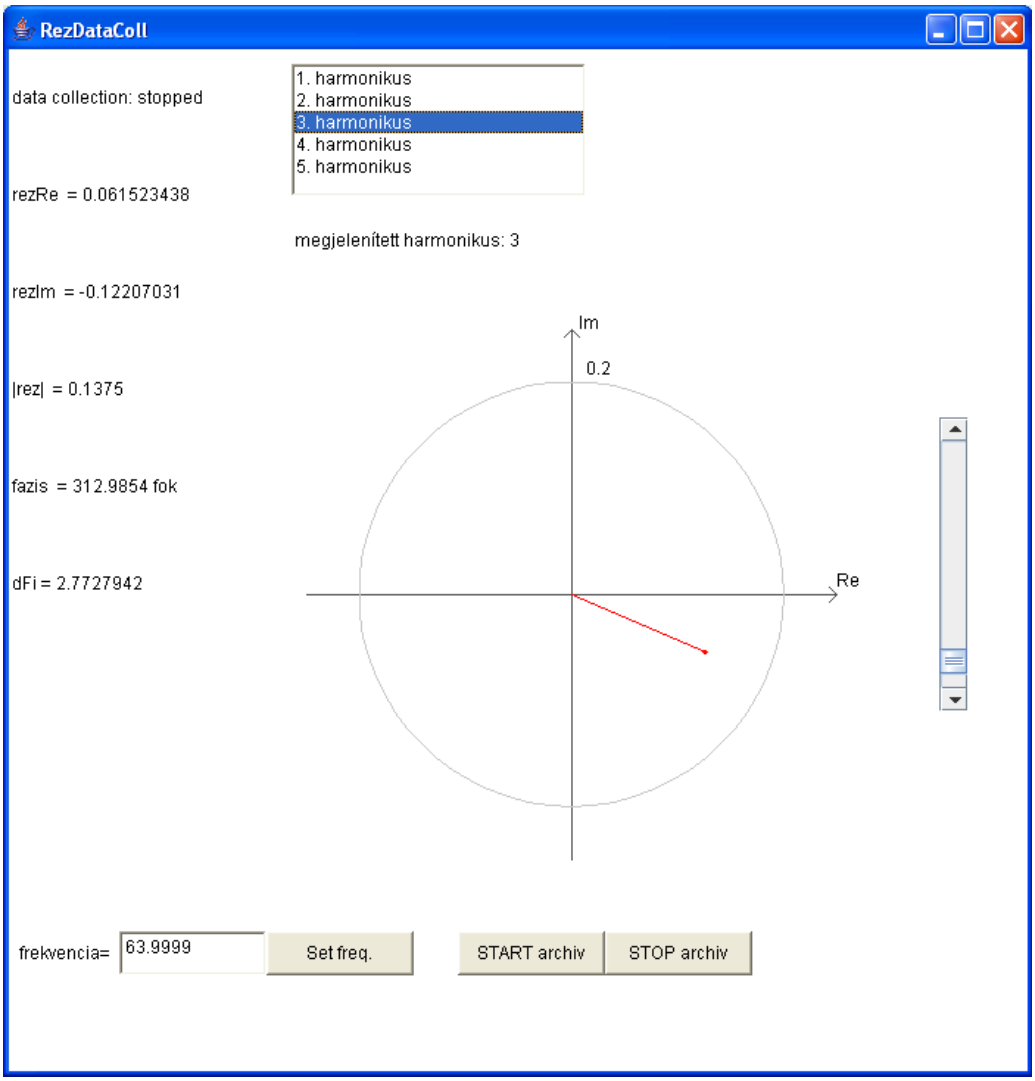


Figure 8: User interface belonging to resonator-based system.

1. It is suitable for displaying the Fourier-coefficients calculated by the sensor: in Figure 8 the vector is marked in red. In the coordinate system, the `Re` axis shows the real part, and `Im` axis represent the imaginary part of the coefficients. The axes can be scaled using the slider next to the coordinate system (right side). The gray circle serves as a reference for estimating the length of the vector.

2. From the list at the top of the window, we can select which Fourier-coefficient we want to display.

3. The Fourier-coefficients are shown numerically on the left side of the window in algebraic and exponential form: `rezRe` and `rezIm` are the real and imaginary parts of the Fourier-coefficient, `|rez|` and `fazis` give the length and phase angle of the Fourier-coefficient.

4. On the left side of the window, `dFi` indicates how much the phase of the fundamental harmonic Fourier-coefficient changed between the transmission times of the two consecutive packets. From this, we can determine how high the frequency error is between the

frequency given to the observer and the frequency of the signal (see the interpretation previously in the description of frequency measurement).

5. The frequency of the observer can be specified in the text field next to the label `frequency=`. The entered value is validated with `ENTER` or by pressing the `Set freq.` button. Then the status of `LED1` on the sensor node changes.

6. The data sent by the nodes can be saved to a file with `START archive` and `STOP archive` buttons. Press the `START archive` button to start saving, and stop it with the `STOP archive` button. After each start, the program overwrites the most recent files, so please save your results.

   After saving the data, run the MATLAB script `preprocessResonatorFiles.m`, which generates files in an easy-to-process format. After running the script, we can use the following files.

   signalFreq.dat: contains the frequency which was set on the user interface. With the help of this file, the frequency associated with the actual measurement can be easily read: `sigFreq = load('signalFreq.dat');`.

   pre_stmpSync.dat, **pre_gwySampFull.dat**: files with the same structure as described in Section 1.2.3.

   pre_sensorResonator.dat: contains the data packets sent by the sensor in the following form:

   `Msg_ID_1` t_1 $\varphi_1(t_1)$ $\Re\{\hat{x}_1(t_1)\}$ ... $\Re\{\hat{x}_5(t_1)\}$ $\Im\{\hat{x}_1(t_1)\}$ ... $\Im\{\hat{x}_5(t_1)\}$ `rezConf=0`

   `Msg_ID_2` t_2 $\varphi_1(t_2)$ $\Re\{\hat{x}_1(t_2)\}$ ... $\Re\{\hat{x}_5(t_2)\}$ $\Im\{\hat{x}_1(t_2)\}$ ... $\Im\{\hat{x}_5(t_2)\}$ `rezConf=0`

   .

   .

   .

   or

   `Msg_ID_1` t_1 $\varphi_1(t_1)$ $\Re\{\hat{x}_1(t_1)\}$ ... $\Re\{\hat{x}_9(t_1)\}$ $\Im\{\hat{x}_1(t_1)\}$ ... $\Im\{\hat{x}_9(t_1)\}$ `rezConf=1`

   `Msg_ID_2` t_2 $\varphi_1(t_2)$ $\Re\{\hat{x}_1(t_2)\}$ ... $\Re\{\hat{x}_9(t_2)\}$ $\Im\{\hat{x}_1(t_2)\}$ ... $\Im\{\hat{x}_9(t_2)\}$ `rezConf=1`

   .

   .

   .

   So the first column contains the ID of the given message: `Msg_ID_r`.

   The second column contains the time instant when the message was sent: t_r.

   The third column contains the phase of the fundamental harmonic basis function in the observer running on the sensor at the time instant when the message was sent: $\varphi_1(t_r)$.

   The `rezConf` flag in the last (14th) column indicates whether the sensor calculates the first five (`rezConf=0`) or the first five *odd* Fourier-coefficients (`rezConf=1`).

   The real part of the Fourier-coefficients are found in columns 4...8: ($\Re\{\hat{x}_i(t_r)\}$). If the first 5 Fourier-coefficients are calculated (`rezConf=0`) these are:

$\left[\Re\{\hat{x}_1(t_r)\},\ \Re\{\hat{x}_2(t_r)\},\ \Re\{\hat{x}_3(t_r)\},\ \Re\{\hat{x}_4(t_r)\},\ \Re\{\hat{x}_5(t_r)\}\right]$; otherwise (`rezConf=1`): $\left[\Re\{\hat{x}_1(t_r)\},\ \Re\{\hat{x}_3(t_r)\},\ \Re\{\hat{x}_5(t_r)\},\ \Re\{\hat{x}_7(t_r)\},\ \Re\{\hat{x}_9(t_r)\}\right]$

The imaginary part of the Fourier-coefficients are found in columns 9...13: ($\Im\{\hat{x}_i(t_r)\}$). If the first 5 Fourier-coefficients are calculated (`rezConf=0`), these are $\left[\Im\{\hat{x}_1(t_r)\},\ \Im\{\hat{x}_2(t_r)\},\ \Im\{\hat{x}_3(t_r)\},\ \Im\{\hat{x}_4(t_r)\},\ \Im\{\hat{x}_5(t_r)\}\right]$; otherwise (`rezConf=1`): $\left[\Im\{\hat{x}_1(t_r)\},\ \Im\{\hat{x}_3(t_r)\},\ \Im\{\hat{x}_5(t_r)\},\ \Im\{\hat{x}_7(t_r)\},\ \Im\{\hat{x}_9(t_r)\}\right]$

Using these data, the complex Fourier-coefficients can be generated: $\hat{x}_i(t_r) = \Re\{\hat{x}_i(t_r)\} + \jmath \cdot \Im\{\hat{x}_i(t_r)\}$.

1.4 Summary of equipement

In this chapter, we summarize the tools used during the measurement.

MITMOT-s We use two MITMOT-s in the system; one serves as a sensor and the other serves as a base station (it is also able to collect data). Both nodes have an IO card (DPY-LED-S-01b), an acoustic sensor card and a 2.4 GHz ZigBee radio card. The acoustic sensor card is capable of detecting acoustic signals and has a direct, unamplified line input to which any analog signal in the range 0 V_{pp} − 3.3 V_{pp} can be fed. The card is AC-coupled, the lower cutoff frequency is found at approximately 20 Hz.

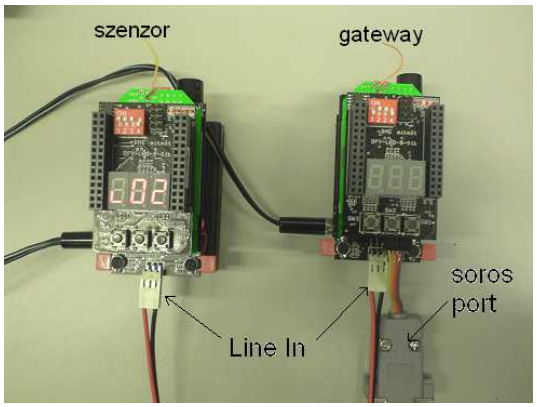


Figure 9: Picture about the MITMOTs

The nodes can be configured during operation using the switches and push buttons. With the help of switch 4 on the DIP switch, we can choose whether the nodes will sample the microphone or the line-in input. The sampling frequency is $f_s = 8 \cdot 10^6/4444 \approx 1800.2$ Hz.

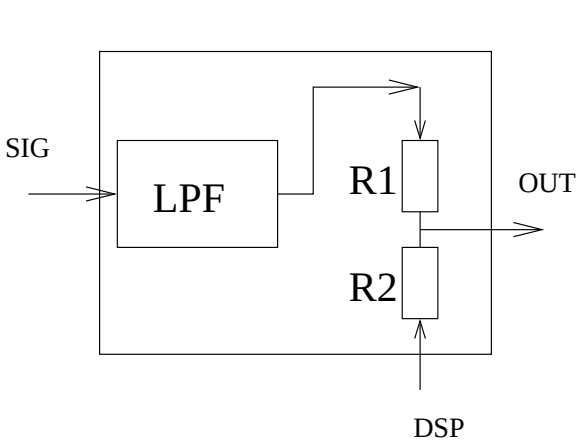
The sensor node can operate in two modes. The operating mode can be set when the node is switched on (or after a reset) by switching between the operating modes with the `SW1` and `SW2` buttons. You can confirm the mode with the `SW3` button. In mode 1, the sensor only transmits the raw samples to the gateway, in mode 2 it calculates the Fourier-coefficients.

If the sensor node calculates the Fourier-coefficients, switch 1 can be used to select whether it calculates harmonics [1...5] or harmonics [1, 3, 5, 7, 9].

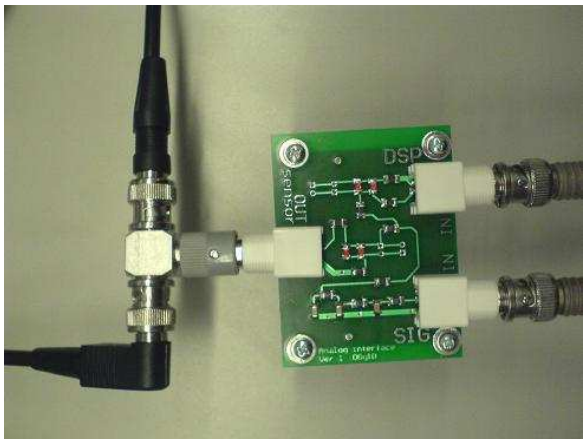
The nodes can be restarted using the RESET button. The RESET button should be used if you want to start the system in a new mode or if you experience abnormal operation.

The state change of LED1 on the sensor node indicates the setting of a new frequency. The frequency can be set using the operator interfaces shown in Figure 8.

Analogue interface board The analog interface card (figure 10) enables the connection between the signal generator, an optional DSP board, and the sensor nodes. On the signal generator side, it includes a simple RC low-pass filter (LPF) to band-limit the signal.



(a) Block diagram of interface board



(b) Picture about the analogue interface board

Figure 10: Analogue interface board

1.5 Measurement tasks

When setting up the measurement arrangements, make sure to first connect the power supply to the devices, and only then connect the signal and data cables. In all cases, we check the position of the switches on the nodes, because they fundamentally affect the operation.

1. Invesitgation of simple data acquisition system

   1.1. Setup the measurement arrangement shown in Figure 4! The sensor node must be started in mode 1. The position of the DIP switches 1...4 on the sensor is as follows: [ON ON ON OFF]. The position of DIP switches 1...4 on the gateway is as follows: [ON ON ON OFF]. On the function generator, it is worth first setting a signal between 40 Hz and 100 Hz, which facilitates the visual evaluation of the signal.

   1.2. Start a new data acquisition by pressing the button `START archive`! The length of the measurement should not be longer than a few sec in the case of the first tests, because processing a large amount of data can be time-consuming. After completing the measurement, run the MATLAB script `preprocessSampleFiles.m`, which converts the saved data into a format that can be easily processed. In addition, files with the prefix 'pre_' must be processed: `pre_stmpSync.dat, pre_gwySampFull.dat, pre_sensorDataFull.dat` (see: page 10)

   Perform the tasks with a 50 Hz and 500 Hz signal. Turn the signal generator output off and on at least once during the measurement in order to investigate transient responses. The amplitude of the signal should preferably fill the measurement range, but in order to avoid overdrive, there should be some safety margin in the system. The results are most informative if we wait for the sensor and base station sampling time instants to "drift away" from each other as much as possible. This occurs when the displayed value of 'mintaveteli kulonbseg' in the data collection program is around 2000 or -2000. Please archive data files manually!

   1.3. Examine the synchronization point pairs based on the `pre_stmpSync.dat` file and plot the local time of the two nodes with respect to each other! Define the timestamp transformation parameters (see (2)), then perform the timestamp transformation based on the defined parameters using the method described on page 5! Plot the time functions of the collected data before and after the timestamp transformation

and evaluate the results! Perform a measurement while continuously changing the amplitude of the signal!

1.4. Perform the time stamp transformation "manually" with the help of LS estimation! (Build the $\mathbf{X}$, $\mathbf{y}$ matrix and vector, then determine the required parameter using formula $\mathbf{p} = \mathbf{X}^T(\mathbf{X}^T \cdot \mathbf{X})^{-1}$.) Modify the sensor synchronization times from zero by 10^6 sec. How does the conditioning of the task change? What is the solution if the offset value of the two units is so different?

1.5. Calculate the measurement results of the sensor at the sampling times of the gateway using linear interpolation (see: (3)) (see MATLAB `interp1`)! Compare the data measured by the two nodes! Increase the frequency of the signal and perform the measurement again! The mean squared deviation can be used as a basis for comparison.

1.6. Perform integer interpolation of the sensor measurement results (including the generation of the time stamp for the new samples that appear due to interpolation), and perform the linear interpolation on these data! Does the method improve the accuracy of interpolation for high frequency signals? Design the interpolating filter for 10x interpolation. Use the `filtfilt` command to filter to eliminate delays!

1.7. Calculate the measurement results of the sensor at the sampling times of the gateway using higher-order interpolation (you can try different orders of polynomials)! Compare the data measured by the two nodes! Increase the frequency of the signal and perform the measurement again! The mean squared deviation can be used as a basis for comparison. The MATLAB `polyfit / polyval` function pair can be used for interpolation. These functions do not perform interpolation on the entire data set, but must be done sample by sample: by stepping through the sampling times (t_i) of the gateway, find the sensor time closest to the given sampling time (n_0 index), then we fit a polynomial of degree $2 \cdot L$ to elements with index $[n_0 - L \dots n_0 + L]$ and evaluate it at time t_i.

2. Investigation of resonator-based system

2.1. Setup the measurement arrangement shown in Figure 4! The sensor node must be started in mode 2. In this case, the sensor transmits the Fourier-coefficients to the PC. The position of the DIP switches 1...4 on the sensor is as follows: [ON ON ON OFF]. The position of DIP switches 1...4 on the gateway is as follows: [ON ON ON OFF]. On the function generator, it is worth first setting a signal between 40 Hz and 100 Hz, which facilitates the visual evaluation of the signal. After the data collection is completed, by calling the `preprocessResonatorFiles.m` script, we can generate the files converted into an easy-to-process format: `pre_stmpSync.dat, pre_gwySampFull.dat, pre_sensorResonator.dat, signalFreq.dat` (see: page 18)

2.2. Set an estimated frequency on the PC user interface (e.g. measured period time on an oscilloscope)! Accurately measure the frequency of the signal using the Fourier-coefficient of the fundamental harmonic as follows:

a) Increase or decrease the frequency by observing the direction of rotation of the fundamental harmonic Fourier-coefficient, until the rotation of the vector representing the Fourier-coefficient stops.

b) By measuring the period time of the rotation of the fundamental harmonic Fourier-coefficient (manually, e.g., using a timer on your mobile phone or wrist watch).

c) Plot the phase of the fundamental harmonic Fourier-coefficient (see: `anlgle` and `unwrap` functions) and fit a straight line to the phase-time function. Using (15), the slope of the line is equal to the circular frequency deviation.

2.3. Start a new data acquisition by pressing the button `START archive`! The length of the measurement should not be longer than a few sec in the case of the first tests, because processing a large amount of data can be time-consuming. Remove the DC component from the gateway data (subtract the average), because it is only in the signal due to the unipolar ADC, but the resonator-based observer does not calculate the DC component. Perform the time stamp transformation (similarly as in the case of the simple data acquisition system) and generate the observed signal from the Fourier-coefficients at the sampling times of the gateway! The reconstruction algorithm can be read on page 16. The reconstruction requires the knowledge of the frequency of the signal, which can be read from the file `signalFreq.dat`. Compare the sampled signal and the signal reconstructed from the Fourier-coefficients. Compare the reconstructed and sampled signal even in the case of signals with different frequencies and varying amplitudes! How can the system follow the amplitude change? During reconstruction, it is worth plotting the time functions of the basis functions $\mathbf{c}_n$ (the real and imaginary part of a complex basis function or the trajectory with the `plot3` function).

2.4. Let's measure how the estimated amplitude changes if you set a different frequency than the real one, e.g. with a frequency error of ± 0.1 Hz, ± 1.0 Hz, ± 2.0 Hz, ± 5.0 Hz, ± 10.0 Hz... Measure the transfer function of the algorithm and plot it! (let the fundamental harmonic frequency be 50 Hz) How the measured transfer function relates to the transfer function of the DFT?

2.5. *Supplementary task*: Check how accurately the time-domain signal can be reconstructed for different frequency estimation errors! (By frequency estimation error, we mean how much the true frequency of the signal differs from the set value.)

2.6. In the system using the resonator observer, calculate the Fourier-coefficients of the signal based on the gateway data and compare them with the data provided by the sensor! See equations (8)-(9)-(10).

2.7. *Supplementary task*: Investigate the error with which a measured signal can be reconstructed if the sensor measures the first five or the first five *odd* Fourier-coefficients! Use a sine, triangle or square signal as a test signal! You can be selected with switch 4 which coefficients the sensor node calculates: (`SW4=ON`: 1...5, `SW4=OFF`: 1,3,5,7,9). This is also indicated by the last column of the saved file, as presented on page 18.

2.8. *Supplementary task*: In the case of the resonator-based data collection system, let's delete a certain percentage of the sensor data, thereby simulating that the sensor transmits data less often! Let's examine the effect of increasingly sparse measurement (both in transient and steady state) on signal reconstruction!