



Comparative Study of Practical Implementation of Time Delay Estimation Methods on Single Board Computer

Vladimir Faerman[✉] , Kirill Voevodin , and Valeriy Avramchuk 

Tomsk State University of Control Systems and Radioelectronics, 40 Lenina Avenue, 634050
Tomsk, Russia
fva@fb.tusur.ru

Abstract. The article discusses the practical implementation of various methods for time delay estimation (TDE) on a Raspberry Pi single-board computer. The relevance of the research is due to the importance of the implementation of TDE methods in the tasks of object positioning and localization. The demand for real-time operation, as well as the requirement to use single-board computers as sensor nodes, imposes high demands on the efficiency of computing. The paper compares various time-domain and frequency-domain TDE methods, including those that utilize a limited set of spectral bins, applicable to problems of localization of acoustic signal sources. The paper considers various methods, their advantages, disadvantages, and computational features. In addition, we have carried out their comparative analysis as well as conducted experimental validation of theoretical estimates of the demands on computing resources. During a series of computational experiments carried out through specially developed software, the computing time and the memory usage are estimated. Based on empirical research on a single-board computer, the Raspberry Pi 4B, we reasonably advise certain methods to be employed in particular scenarios for localization of an acoustic source in space using the Raspberry Pi single boards.

Keywords: Time Delay Estimation · Raspberry Pi 4 · Fast Fourier Transform · Goertzel Algorithm · Computational Grids · Sliding Discrete Fourier Transform

1 Introduction

The significance of effective methods for time delay estimation (TDE) lies primarily in their wide practical application in local positioning systems [1, 2]. Schematically, the problem of local positioning is presented in Fig. 1 [1]. Depending on the specific circumstances within the scenario, positioning tasks can be considered as passive or active. In passive tasks, the position of the object that is the direct emitter of the signal is determined. In active tasks, the mobile object being positioned reflects a dedicated signal emitted by the locator [3]. It is also possible to reverse the composition of the system shown in Fig. 1, where the mobile object will be considered as the signal receiver. In turn, stationary nodes will become transmitters [4]. It should be noted that the reverse passive

scenario is not significantly different from the regular passive one when considering solely the TDE problem. Time delay estimation methods are normally applied to all those scenarios [1].

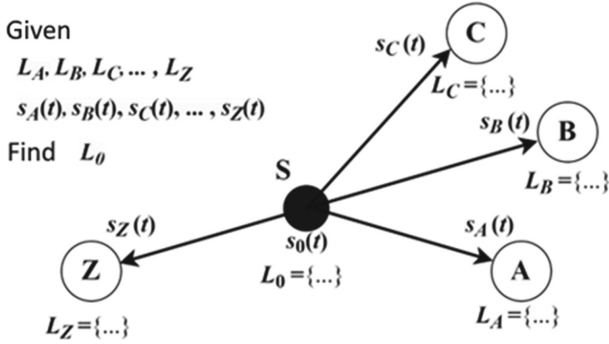


Fig. 1. Generalized TDE scheme (passive non-reverse scenario): S – mobile object (active emitter); A, B, C, \dots, Z – array of stationary sensors; $s_0(t)$ – signal emitted by the source S ; $L_A, L_B, L_C, \dots, L_Z$ – exact positions of the corresponding sensors; $s_A(t), s_B(t), s_C(t), \dots, s_Z(t)$ – signals received by corresponding sensors; L_0 – position of the source S .

The high demand for object positioning systems, in turn, is associated with the development of wireless technologies and advances in industrial automation. Recent development of the Internet of Things [5, 6] and the spread of smart devices [7, 8] have opened a great new field of application of positioning systems. The gradual introduction of such devices and systems into the consumer goods market makes it relevant to further reduce their cost. The principal factor of cheapening in this case is the reduction in the cost of hardware for smart devices or sensor nodes [6]. Systems on a chip (SoC) and SoC based single board computers are often considered as core computing units for positioning systems. The advantage of the latter is the developed peripherals and wide availability of hardware expansion modules, sufficient to solve most user tasks [9].

The main reason for choosing single-board computers is their high portability and self-sufficiency, as well as low power consumption and reduced cost compared to personal computers. However, you have to trade this for worse performance characteristics and lower volume of available RAM. This has to be considered when implementing digital signal processing algorithms for practical applications.

In particular, a specific feature that must be taken into account when implementing local positioning techniques is the limited performance of the hardware platform, especially when operating in real time. This makes it relevant to study the practical methods of TDE and variants of their practical implementation in relation to various scenarios of application. The expected result is the reasonable choice of the TDE methods and their variants applicable to such scenarios as real time positioning of a mobile object, positioning of a signal source with *a priori* known spectral composition, and some others.

This article presents a theoretical and empirical comparison of different implementations of time-domain and frequency-domain methods of TDE in terms of computational

performance and the amount of memory involved. In the experimental part of the work, a Raspberry Pi 4B single board computer was used as a hardware for tests.

2 Overview of TDE Methods

Time delay estimation is the most common technique applied to the local positioning problem. The essence of TDE is to measure a time lag between signals received by an array of spatially distributed sensors. In the basic passive scenario, the source of those signal is a positioning object.

For the sake of certainty, let us assume that only two sensors are used. In this case, the signals received by the sensors are described by the following expression [10]

$$\begin{aligned} s_a(t) &= K_a \cdot s_0(t - \tau_a) + n_a(t), \\ s_b(t) &= K_b \cdot s_0(t - \tau_b) + n_b(t), \end{aligned} \quad (1)$$

where s_0 is the signal emitted by the object; K_a, K_b are the attenuation coefficients of signals along the propagation path; τ_a, τ_b are the delays between the received and the emitted signals; n_a, n_b are additive noises, s_a, s_b are received signals. So the required lag time is expressed as

$$\tau_{ab} = (t - \tau_a) - (t - \tau_b) = \tau_b - \tau_a. \quad (2)$$

The value τ_{ab} is often referred to as the time difference of arrival (TDOA). Wide class of positioning methods is based on the analysis of set of TDOA estimations [1]. Depending on the practical setting, they can be used to determine the coordinates of an object by the method of multilateration in linear coordinates [11–13], in plane coordinates [14, 15], or in three-dimensions [5, 8, 16]. Such problems are quite typical for such fields of science and technology as local positioning and indoor navigation, wide area positioning of mobile subscribers in communication networks [5, 8], locating of pipeline leaks with leak noise correlators [11–13].

The applied methods to the aforementioned problem can be divided into two groups: time and frequency [17]. This segregation of methods is based on differences in the form of representation of signals directly in their analysis.

2.1 Time-Domain Methods

Time-domain TDE methods usually assume analyzing the correlogram [3]. The lag time can be obtained as an argument of the correlation function at which it reaches its maximum value:

$$\tau_{ab} = \arg \max (R_{ab}(\tau)), \quad (3)$$

where $\arg \max$ is the operator for obtaining the argument value at which function is maximized; $R_{ab}(\tau)$ is the correlation function of s_a and s_b .

The conventional way of calculating the correlation function of sampled signals $s_a(t_i)$ and $s_b(t_i)$ applies the convolution theorem [11].

$$R_{ab}(\tau_j) = F^{-1}(F^*(s_a(t_i)) \times F(s_b(t_i))), \quad (4)$$

where F is operator of discrete Fourier transform (DFT) operator; F^{-1} is operator of inverse DFT (IDFT); $*$ denotes unary operation of element-wise complex conjugation; \times denotes binary operation of element-wise product.

Despite the prevalence of the approach based on (3) and (4), the practical implementation of time-domain methods may vary in a sensible way. The variants differ mainly by the algorithm that is used to perform DFT and IDFT operations.

2.2 Frequency-Domain Methods

Frequency-domain methods extract the lag time directly from the cross-spectrum of the analyzed signals. The discrete spectra $S_a(f_k)$ and $S_b(f_k)$ of the sampled signals $s_a(t_i)$ and $s_b(t_i)$ are complex valued, therefore we can consider them as following [18]

$$S_{a,b}(f_k) = F(s_{a,b}(t_i)), \quad S_{a,b}(f_k) = X_{a,b}(f_k) \times \Phi_{a,b}(f_k), \quad (5)$$

where $X_a(f_k)$, $X_b(f_k)$ are amplitude spectra; $\Phi_a(f_k)$, $\Phi_b(f_k)$ are phase spectra. The amplitude spectrum carries information about the energy properties of the signal. The phase spectrum carries information about the temporal features of the signal, in particular, the time shift is reflected in it.

The cross-spectrum, respectively, has the form

$$S_{ab}(f_k) = S_a^*(f_k) \times S_b(f_k),$$

$$S_{ab}(f_k) = X_{ab}(f_k) \times \Phi_{ab}(f_k),$$

$$X_{ab}(f_k) = X_a(f_k) \times X_b(f_k), \quad \Phi_{ab}(f_k) = \Phi_b(f_k) - \Phi_a(f_k). \quad (6)$$

The phase component of the cross-spectrum $\Phi_{ab}(f_k)$ is used to extract information about TDOA. The following formula can be used directly to estimate the lag [18]

$$\tau_{ab} = \sum_k \Theta_{ab}(f_k) \cdot f_k \bigg/ 2\pi \sum_k f_k^2, \quad (7)$$

where $\Theta_{ab}(f_k) = U[\Phi_{ab}(f_k)]$ is the result of applying the unwrapping operator U to $\Phi_{ab}(f_k)$. [18]

Expression (7) is established based on the full spectral representation of the signals. Taking into account the equivalence of the information contained in the time and frequency representations of the signal, it is correct to consider (7) as an analogue of (3). The mathematical identity of these methods in terms of the potentially achievable accuracy is shown in [19]. However, it should be noted that in practical cases, they give different results. This is due to both the difference between real signals and model signals, and the inevitable differences in their computational simulation [20].

In [20], an alternative variant (7) is proposed, which allows one to use an arbitrary set of samples of the phase cross-spectrum. This makes it possible to apply the frequency method in situations where noise prevails at low frequencies. The use of an alternative formula also makes it possible to determine the time lag without computing the entire spectrum. This feature is discussed in detail in the following section.

2.3 Signal Processing in Practical TDE

In practical cases, the basic methods of TDE have low accuracy due to contamination of the source signal by additive noise on the side of the receiving sensors. Reduction in the negative impact of noise can be achieved by averaging estimates of the spectral characteristics of signals. Each spectral estimate is obtained by the short-time Fourier transform method [10]. In general, the time windows at the input of the transformation can overlap and have a shared subset of signal samples. The expressions for computing the correlation function for (3) and the phase cross-spectrum for (7) will respectively take the following form:

$$R_{ab}(\tau_j) = F^{-1} \left(\frac{1}{Q} \cdot \sum_q \left[F^* \left(s_a^{(q)}(t_i) \right) \times F \left(s_b^{(q)}(t_i) \right) \right] \right), \quad (8)$$

$$\Theta_{ab}(f_k) = U \left[\frac{1}{Q} \sum_q \left[\Phi \left(s_b^{(q)}(t_i) \right) - \Phi \left(s_a^{(q)}(t_i) \right) \right] \right],$$

$$\Theta_{ab}(f_k) = U \left[\arg \left(\frac{1}{Q} \sum_q \left[F^* \left(s_a^{(q)}(t_i) \right) \times F \left(s_b^{(q)}(t_i) \right) \right] \right) \right], \quad (9)$$

where Q is the total number of time windows at the input of DFT; \arg is the operator that returns argument of a complex number; $s_a^{(q)}$, $s_b^{(q)}$ are subsets of signal samples belonging to the time window with index q .

In addition to averaging spectral estimates, frequency-weight functions of the form $\Psi_{ab}(f_k)$ are used to further reduce the influence of noise [21]. The values of the samples of the frequency-weight functions are positive and do not exceed unity. Values $\Psi_{ab}(f_k)$ close to unity indicate that at this frequency bin f_k , the signal overall prevails over noise throughout the entire observation period. Frequency-weight functions can be used with both time-domain [11, 21, 22] and frequency-domain methods [12, 23] of TDE. Despite the variety of such functions, averaged spectral estimates are always used to get them. From a computational standpoint, obtaining additional spectral estimates does not differ from the cross-spectrum estimate used in (8). For this reason, weighing in the frequency domain is not considered in the course of the further experimental study.

2.4 Variants of Fourier Transform Implementation

As shown above, both time and frequency methods of TDE require spectral transformations. In the practice of digital signal processing, DFT algorithms are used for this. Some of the most effective solutions are classified as fast Fourier transforms (FFT). Among the latter, the Coolie-Tukey algorithm is the most well-known and widespread [24].

Despite the significant computational advantages of FFT over the straight computation of the DFT, its use has a number of inconvenient features. Firstly, most FFT algorithms impose restrictions on the number of samples at the input of the transform. Secondly, those algorithms allow only the computation of the entire spectrum. This is

redundant in cases where the informative signal is localized in several *a priori* known frequency bins. Thirdly, the obtaining of new data (a forward shift of the time window by several samples) requires a full-fledged application of the FFT to obtain a new spectral estimate. This complicates the use of FFT when operating in real time. The use of small windows is not always acceptable, since the size of the time window is associated with frequency resolution and noise tolerance. In contrast, the use of large windows, in combination with calling the transformation every time new data arrives, creates a large computational load.

Special DFT implementations were proposed for all the cases described above, where the FFT is limited in application. In particular, the chirp Z-transform (CZT) makes it possible to obtain an arbitrary number M of spectral bins using time windows composed of an arbitrary number N of ticks [25]. It should be noted that this limitation of the FFT is not essential for applying to TDE problem so we do not consider CZT in further study.

The Goertzel algorithm was initially proposed to compute individual frequency bins within the signal spectrum [26]. The use of this algorithm in conjunction with the frequency-domain methods of TDE allows one to obtain a time lag without computing the entire spectrum. This feature gives a computational advantage in some TDE scenarios, and therefore we will investigate it further.

A recursive sliding DFT algorithm can be used to obtain spectral estimates in real time [27]. The advantage of this algorithm is the ability to reevaluate the already available spectral characteristics based on newly received data. We will further investigate the performance of SDFT and the corresponding amount of used memory to determine its possibility of application to TDE problem.

It should be noted that by the moment numerous different recursive DFT algorithms have been developed and described [28, 29], which remained beyond the framework of this study. However, the potential of applying a few of them to TDE problem is discussed in conclusion.

3 Computational Study

To determine the operational capabilities of a sensor node based on a single-board computer, a series of computational experiments was carried out. During the experiments, an array of dummy data was processed with time-domain and frequency-domain TDE methods described in the previous section.

Further in this section we present estimates of the computational performance and memory usage benchmarks related to the most critical stages of the implementation of the considered methods. The discussion section showcases a comparison between the empirical outcomes derived from empirical investigations and theoretical estimations. Operation limits for the TDE device that are implied from the study are also could be found within the discussion.

3.1 Raspberry Pi 4B Hardware

Computational experiments feature a Raspberry Pi 4B single-board computer [30] with a HiFiBerry DAC + ADC Pro expansion board [31] shown in Fig. 2. The Broadcom

BCM 2711 SoC is the core processing unit of the computer. This SoC incorporates a quad-core general-purpose processor with the Cortex-A72 microarchitecture and a VC6 graphics core, along with some peripheral components. The HiFiBerry sound card was used exclusively in some segregated tests to verify the operability in real time for specific input data rates and particular preset of computational parameters. Therefore, its characteristics are not significant in the context of this study.



Fig. 2. Raspberry Pi 4B with HiFiBerry DAC + ADC Pro sound card attached on top.

3.2 Testing Software

For the purpose of this study, we have developed software for automated experimentation and statistical preprocessing of acquired data. We elected C++ as the main programming language, which was used to unify the program interfaces and implement wrap distinct computational functions.

Performance critical software components were implemented in low level in C. Our algorithmic implementation of the TDE methods largely corresponds to the description given in Sect. 2. To implement the FFT, we have used the current version of the FFTW library, that is in fact commonly considered as the branch standard. We implemented software components for SDF and Goertzel transform in a low level based on the algorithms described in [26] and [27] respectively.

We implemented a special class dedicated to acquisition of statistical data on computation time. Raw time benchmarks were gathered on calls of execution method of wrapper class. Each benchmark was reiterated 150 times. Then, the raw data underwent statistical processing. For each benchmark, we recorded the minimum and maximum execution times, the average time, as well as the standard deviation of time. Sample code for gathering and processing benchmarks is shown in Fig. 3.

Only dynamic allocation was taken into account when evaluating memory usage. This is due to the fact that a fair share of memory usage is associated with storing in buffers time series, complex spectra and precomputed constants for transforms. Due to

```

#ifdef FFT_GOERZEL_NUMBER
std::cout << "Goerzel freq number benchmark" << std::endl << std::endl;
for( auto N_ : N )
{
    std::vector< double > data( 1 << N_ );
    for( auto& sample : data )
    {
        sample = static_cast<double>( std::rand() ) / RAND_MAX - 0.5;
    }
    for( auto diff_ : diff )
    {
        transform::cpu::forward::Goerzel transform( 1 << N_, GRZ_INDEX_FIRST, GRZ_INDEX_FIRST + diff_ );
        transform.SetReal( data );
        for( size_t i = 0; i < NUM_BENCH_ITERS; ++i )
        {
            auto begin = std::chrono::steady_clock::now();
            transform.Execute();
            auto finish = std::chrono::steady_clock::now();
            auto duration = std::chrono::duration_cast< std::chrono::microseconds >( finish - begin).count();
            bench.UpdateRes( duration );
        }
        bench.CalcStats();
        std::cout << "N: " << N_ << "\t" << "diff: " << diff_ << " \t\t"
            << "min: " << bench.GetMin() << " \t" << "max: " << bench.GetMax() << " \t"
            << "mid: " << bench.GetMiddle() << "\t" << "CKO: " << bench.GetCKO() << std::endl;
        bench.Reset();
    }
    std::cout << std::endl;
}
std::cout << std::endl;
#endif // FFT_GOERZEL_NUMBER

```

Fig. 3. Visual Studio screenshot that shows implementation of time measurements.

their size, these data arrays have to be stored in dynamic memory. Such an approach to the evaluation of memory usage is tolerant to distortion by memory, that is used on the stack and not directly related to the algorithms under study. The influence of the latter could not be excluded if the entire memory associated with the process was used as an estimate.

The *Valgrind* software was used to collect data on the allocated memory [32]. This tool is a specialized memory management service, debug utility system and profiler for software developers. Its functions include but not limited to the search for memory leaks, register attempts to accesses beyond the boundaries of allocated areas or use of uninitialized memory, and the investigation of other memory-related bugs.

3.3 Estimation of Computation Time

The variant of DFT implementation heavily influences the performance of a TDE method. This follows from (3) and (8), as well as (7) and (9), which is coherent with acquired experimental data. Any of these TDE methods requires at least $2 \cdot Q$ DFTs. This computational operation significantly prevails in (9). Other operations are mostly computationally simple: element-wise products of complex values, a unitary element-wise taking argument of complex numbers and element-wise multiplication by a scalar value. On the other hand, (8), in addition to similar element-wise operations, requires a single execution of the IDFT, which is computationally equivalent to an additional forward DFT.

For this reason, we further provide runtime estimates related only to the implementation variants of DFT. The execution time of the rest of the operations is not of comparable interest, because it has an auxiliary effect on the performance of the TDE methods, and also usually depends on the size of the time window N linearly.

The estimations of FFT execution time for various sizes N of the time window are presented in Table 1. Here and further, the following designations are used: T_{min} – minimum computation time; T_{max} – maximum computation time; T_{ave} – mean computation time; ΔT – standard deviation (half width) of computation time. Since many random factors can negatively affect the calculation time, we chose the minimum time as the most reliable estimate for the purpose of performance comparison. Key benchmarks for FFT are shown in Fig. 4.

When estimating the execution time of the Goertzel transform, we have varied both the number of samples in time windows and the number of calculated frequency bins. Since a theoretically predicted linear dependence of the execution time on the number of frequency bins presented in all experiments, in Table 2 we showed the computation time for a single frequency bin. Key benchmarks for Goertzel algorithm of DFT are shown in Fig. 5.

Similarly, when estimating the execution time of the SDFT, we varied the number of samples in time windows as well as the overlap rate between adjacent windows. Since we predictably found a linear dependence of the computation time on the number of newly introduced time samples in the previous time window, we elected to present the computation time for a single sample in Table 3. Key benchmarks for SDFT are shown in Fig. 6.

3.4 Estimation of Memory Usage

Estimates for the memory usage are given only for DFT variants, for similar reasons. However, the memory requirements depend on a TDE method to a greater extent than the performance. For instance, the use of frequency weighting functions requires storing in memory several additional spectral estimates (usually power spectra) as well as a set of frequency coefficients. So time-domain methods require the storage of whole spectra and the full set of frequency coefficients, while frequency-domain methods can rely on a limited set of frequency samples that require less memory to store.

Empirical estimates of the memory usage are presented in Fig. 7. The results of the study indicate the slight superiority of the Goertzel transform in this aspect. The actual advantage of the latter may be higher, given that the volume of required memory is dependent on the number of computed frequency bins (see Fig. 8). However, if we elect not to preserve inputs with FFT we can even make memory its usage lesser than Goertzel for full spectrum case.

Figure 8 clearly shows that the memory required for Goertzel transform is linearly dependent on the number of bins that have to be calculated. The constant term in the linear equation tends to become less significant with the size of the time window.

Table 1. Time to compute full spectrum with FFT.

N , samples	T_{min} , mcsec	T_{max} , mcsec	T_{ave} , mcsec	ΔT , mcsec
256	3	13	3.080	0.823
512	6	14	6.053	0.651
1024	13	19	13.040	0.488
2048	29	48	29.147	1.551
4096	76	129	77.853	6.912
8192	197	299	204.227	17.033
16384	428	639	444.313	39.560
32768	1006	1522	1046.770	82.995
65536	2902	3500	2969.608	76.057
131072	8715	11136	9101.190	426.446

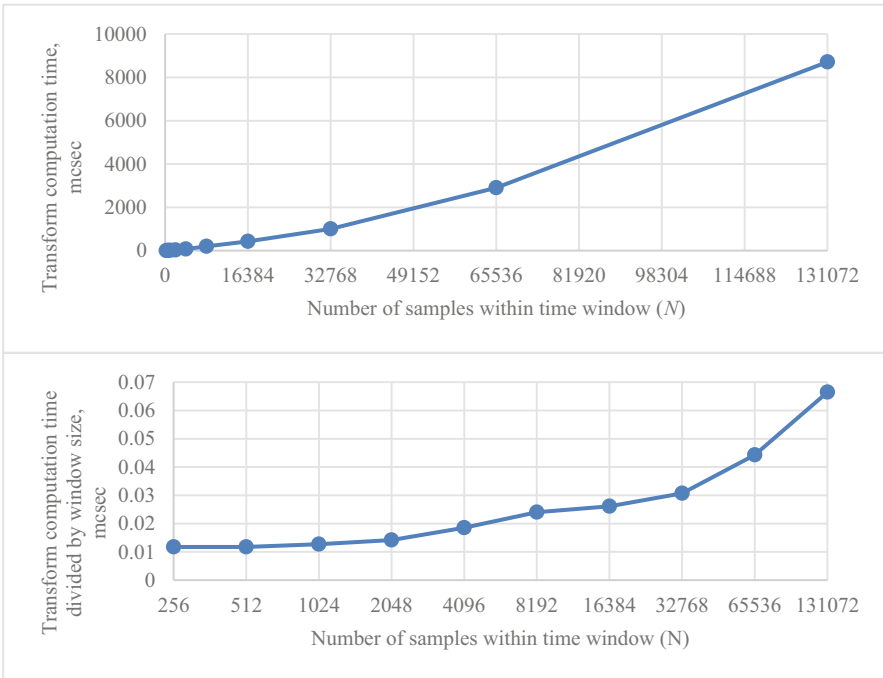


Fig. 4. Computation time vs number of samples within time window for FFT: for a full input window of N samples (on top); for a single input sample (on bottom).

Table 2. Time to compute one frequency bin with Goertzel transform.

N , samples	T_{min} , msec	T_{max} , msec	T_{ave} , msec	ΔT , msec
256	2	3	2.188	0.035
512	2	5	3.089	0.495
1024	5	6	5.753	0.032
2048	11	12	11.478	0.014
4096	22	24	22.756	0.110
8192	46	47	46.482	0.067
16384	93	95	93.306	0.111
32768	186	188	186.589	0.112
65536	373	374	373.163	0.030
131072	747	749	747.767	0.210

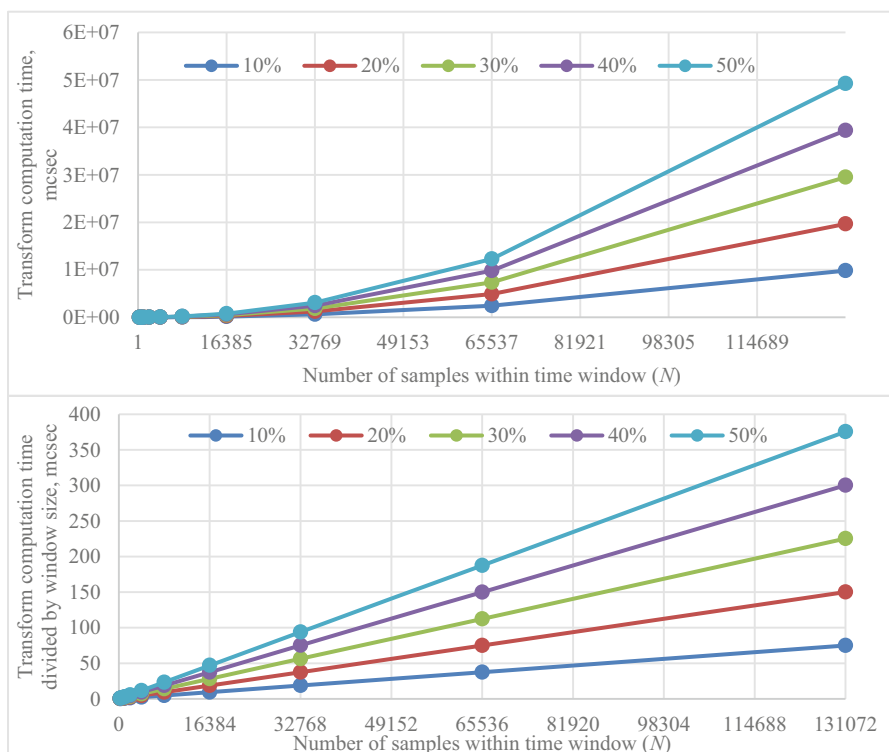
**Fig. 5.** Computation time vs number of samples within time window for Goertzel transform (various rates of computed frequency bins are indicated by the color of a curve): for a full input window of N samples (on top); for a single input sample (on bottom).

Table 3. Time to compute SDFT with almost overlapping time windows and precomputed spectrum for previous window (all samples but one are in both time windows).

N , samples	T_{min} , msec	T_{max} , msec	T_{ave} , msec	ΔT , msec
256	1	6	3.498	0.914
512	3	6	3.69	0.204
1024	7	9	7.277	0.11
2048	14	20	14.687	0.452
4096	29	30	29.551	0.039
8192	57	59	58.11	0.087
16384	115	117	115.945	0.117
32768	231	233	231.93	0.126
65536	487	491	488.799	0.692
131072	1085	1093	1088.615	1.269

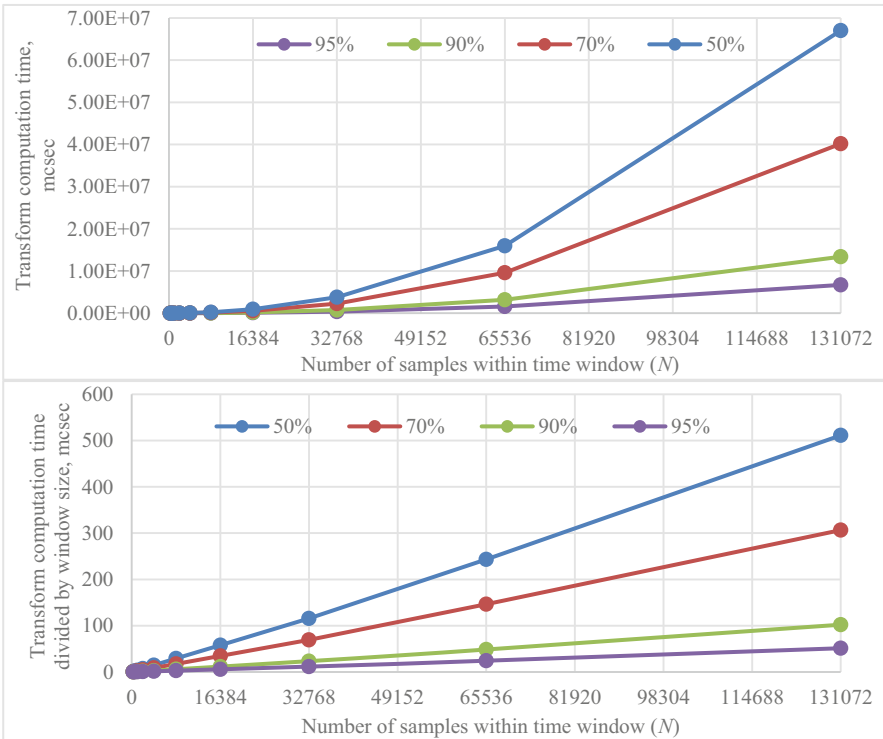


Fig. 6. Computation time vs number of samples within time window for SDFT (various rates of overlapping samples are indicated by the color of a curve): for a full input window of N samples (on top); for a single input sample (on bottom).

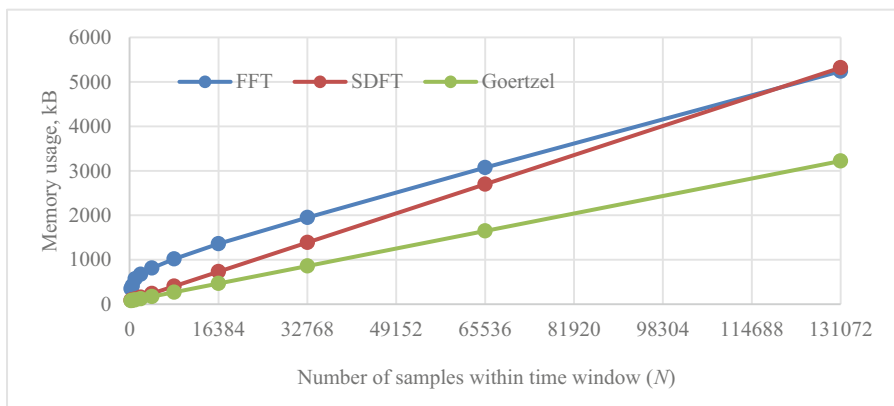


Fig. 7. Memory usage vs number of samples within time window for all considered DFT variants.

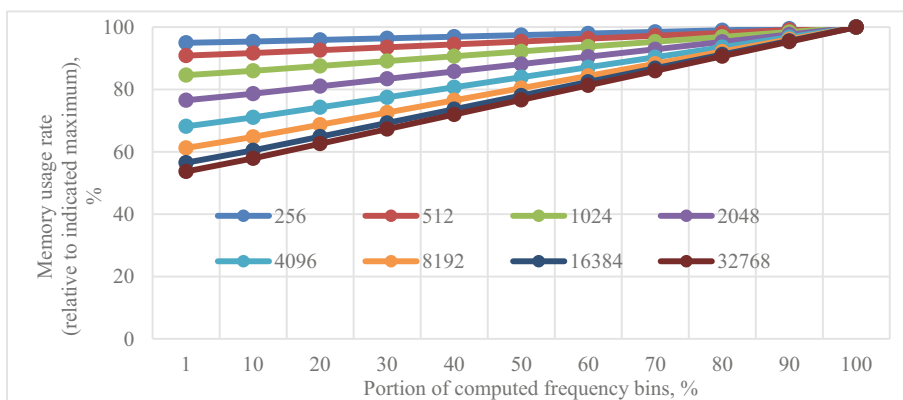


Fig. 8. Memory usage rate (compared to maximum value used for computation of full spectrum) vs rate of computed frequency bins for Goertzel transform (various windows size are indicated by the color of a lines).

4 Discussion

Our empirical results correspond well to theoretical estimations of complexity in regard to memory usage and computational operations required. Calculating the DFT for a real input time series using the FFT requires $(N/2) \cdot \log_2(N)$ complex multiplications and $N \cdot \log_2(N)$ complex additions. Each complex multiplication is composed of 4 real multiplications and 2 real additions. A complex addition is composed of 2 real additions. Thus, computing a spectrum via applying FFT to a time window of N samples requires $2N \cdot \log_2(N)$ real multiplications and $3N \cdot \log_2(N)$ real additions. The asymptotic computational complexity of the transform is $O(N) = N \cdot \log_2(N)$ and it is consistent with Fig. 4.

Each recursive FFT call requires splitting and reordering the interim results obtained at the current step of recursion. In the proposed implementation, a separate buffer was

allocated for spectral data. However, it is possible to save about one third of memory by overwriting the initial sequence during computations. However, it is necessary to store the pre-calculated rotation multipliers prior to the transform, or performance will be compromised. Asymptotically, the memory usage of the FFT is $O(N) = N$, which is consistent with Fig. 7.

The Goertzel algorithm for calculating K frequency bins for an input time series of N samples requires $K \cdot 4N$ real multiplications and $K \cdot 5N$ real additions. Thus, its asymptotic computational complexity is $O(K, N) = K \cdot N$. Full scale real-valued DFT by Goertzel algorithm ($K = N/2 + 1$) is inefficient and requires $2(N^2 + 2N)$ real multiplications and $5(N^2/2 + N)$ real additions. These estimates correspond well to those curves presented in Fig. 5. The implementation of the Goertzel algorithm requires storing K precomputed complex rotation multipliers. At the same time, it is also necessary to store the input series of N real samples as well as K computed output spectral estimates. The asymptotic memory requirement of the Goertzel algorithm is $O(N, K) = N + K$, which is consistent with Fig. 8.

The recursive SDFT algorithm relies on the already available spectral estimates when recomputing spectrum with the arrival of new input data. Processing each new time sample requires $N + 2$ real multiplications and $N + 2$ real additions. The asymptotic complexity of the transformation is $O(M, N) = M \cdot N$, where M is the number of newly arrived non-processed time samples. Processing a full time window of N samples requires $N^2 + 2N$ additions and $N^2 + 2N$ multiplications. These estimates are consistent with the curves shown in Fig. 6. Like other variants, the sliding transform utilizes an array of rotation multipliers as well as buffers to store input and output sequences. The difference of SDFT is that the input series may not comprise a complete time window of all N samples in a first place. However, an additional internal buffer is required to store the N time samples which were used to obtain current spectral estimates. Even though SDFT can be called with any number of samples as input, it must be at least N samples in total before the first spectral estimate is produced. The requirement for an additional buffer leads to the fact that SDFT slightly underperforms in the aspect of memory requirements. That can be seen in Fig. 8. The asymptotic memory requirement of SDFT is $O(M, N) = M + N$.

Comparison of DFT implementation variants has shown that FFT is suitable in a wide range of scenarios, with few rare exceptions. Figure 9 shows the range of parameters of a computational problem in which Goertzel algorithms outperforms FFT. As far as a frequency-domain TDE method requires at least three frequency bins to draw a regression line, computational advantages can be achieved only by large time window sizes. This results in high frequency resolution. So to be practical in conjunction with a frequency-domain TDE method, SDFT requires an accurate *a priori* knowledge of the frequency localization of the signal as well as the absence of scattering during its propagation.

Figure 10 shows the range of parameters of a computational problem when SDFT has an advantage over FFT in execution time. One can infer from the figure that the use of the recursive algorithm is advisable only if an exceptionally high rate of spectrum recalculation is required. The sampling frequency is usually 44100 Hz if we assume a problem of positioning a mobile object via an acoustic channel. So in this case, the use of SDF will be practical only when the position of the object (along with spectral estimates) need to be updated at a rate exceeding 5000 Hz. Such a scenario seems not to be very realistic.

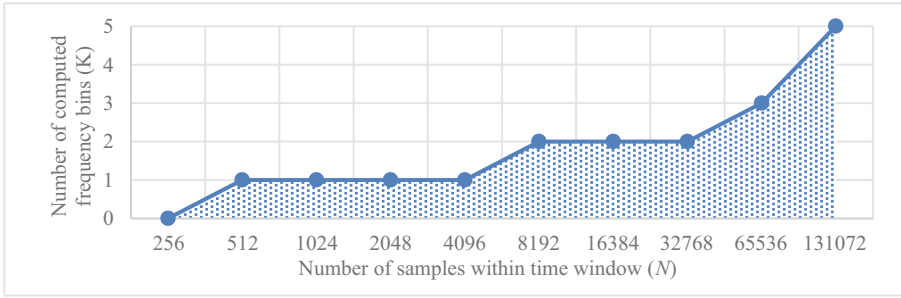


Fig. 9. Area in the domain of computational parameters when Goertzel outperforms FFT.

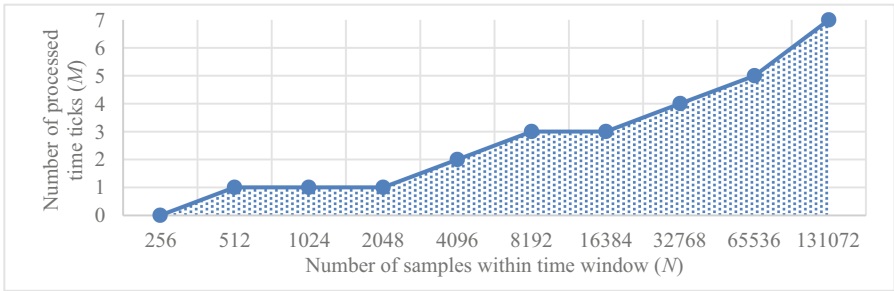


Fig. 10. Area in the domain of computational parameters when SDFT outperforms FFT.

A comparison of DFT implementation variants in the aspect of memory requirements showed that the Goertzel algorithm has a slight advantage. However, the amount of memory used is generally comparable for all variants, and this advantage appears to be of not high practical importance. If a critical limitation on memory is a case, then it is possible to save about one third of memory used by FFT just by giving up preserving input.

In the course of further discourse, we will assume that the functions of the sensor node are reduced to receiving continuously incoming signals, buffering them, processing them and output the results of their processing. Let us also assume that the intensity of data rate remains unaltered through all operating session, and the processing of the obtained results and their output are carried out asynchronously. The similar situation is described and modeled in [33].

Hence, real-time operating can be attributed to two parallel processes:

- processing of incoming data and refinement of spectral estimates (usually by coherent averaging of instantaneous spectra);
- and utilizing those spectra to measure time lags with TDE methods, use time lags to estimate object position solving multilateration problem.

The second process is not hardly synced with the first one and can be performed on demand or on a residual basis. On the contrary, acquiring and processing of incoming data should be done as soon as it arrives in order to avoid buffer overflow and data loss.

Let us denote the intensity of the incoming data flow as B and define as

$$B = f_d \cdot n,$$

where f_d – sampling rate; n – number of channels. So, the total number of time windows Q of size N that need to be processed during time period T_0 can be defined as

$$Q = T_0 \cdot \frac{B}{N} = T_0 \cdot \frac{f_d \cdot n}{N \cdot (1 - s)},$$

where s – overlap rate between adjacent time windows ($0 \leq s < 1$). By supposing that the processing time of a window is predominantly determined by the DFT computation time, we estimate the total time T_Q takes to process all Q windows:

$$T_Q = Q \cdot \frac{T(N)}{k(N)} \cdot \frac{B}{N} = T_0 \cdot \frac{f_d \cdot n \cdot T(N)}{N \cdot (1 - s) \cdot k(N)},$$

where $T(N)$ – average computation time of DFT; k – the share of DFT in the total computational load. The ratio T_Q to T_0 is the fraction ρ of time that the machine spends on processing the input data stream:

$$\rho = \frac{T_Q}{T_0} = \frac{f_d \cdot n \cdot T(N)}{N \cdot (1 - s) \cdot k(N)}.$$

Let take that about two-thirds of computing resources need to be reserved in order to ensure timely and regular calculation and display of the object's position. Therefore, the number of channels that a computing device can serve can be roughly estimated as

$$n \leq \frac{N \cdot (1 - s) \cdot k(N)}{3 \cdot f_d \cdot T(N)}.$$

It is safe to assume that $k(N) \geq 0.66$ for any $N \geq 256$. We have previously determined the empirical values of $T(N)$ for the FFT and presented them in Table 1. A rough estimate using the formula above shows that the Raspberry Pi 4B is capable of acquiring and processing sound data from at least 16 channels at a frequency of 44100 Hz or from 4 channels at a frequency of 192000 Hz with an overlap rate of 75% in both cases. This potentially makes it possible to reevaluate the position of an object dozens of times per second, even using reasonably large size windows (for example, $N = 16384$). These qualitative estimates are confirmed empirically during test runs of Raspberry Pi 4 with a HiFiBerry module. More detailed and accurate benchmarks are planned for the future.

5 Conclusion

In this work, a comparative study of various implementations of TDE methods was made in respect of applicability on a single-board computer Raspberry Pi 4B. In the course of theoretical study of the issue and empirical research, it was established that DFT is the most computationally demanding operation, and it in a large extend determines the performance of the methods.

A comparison of various widely known DFT algorithms, in particular FFT, the Goertzel algorithm and the recursive SDFT algorithm showed significant advantages of FFT in performance and their equivalence in memory usage. Despite the fact that some areas of computational parameters in which the Goertzel algorithm and SDFT outperform the FFT, it is of little practical value.

The Raspberry Pi 4B single-board computer has sufficient computational capabilities to be used for positioning objects via an acoustic channel. The computer is able to process data streamed via 4 (or more) acoustical channels and to compute spectral estimates in a soft real time at a rate of at least 10 times per second. This way, FFT algorithms can be effectively utilized with time-domain and frequency-domain TDE methods.

In the future, other special DFT algorithms should be checked as well. In particular, the sliding Goertzel algorithm [34–36], which combines the features of both SDFT and the classical Goertzel algorithm, can probably make a competition to FFT in the scenario of tracking a mobile object in real-time.

References

1. So, H.C.: Source localization: algorithms and analysis. In: Position Location: Theory, Practice, and Advances, pp. 25–66 (2011). <https://doi.org/10.1002/9781118104750.ch2>
2. Gustafsson, F., Gunnarsson, F.: Positioning using time-difference of arrival measurements. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing – Proceedings, pp. 553–556. IEEE (2003). <https://doi.org/10.1109/icassp.2003.1201741>
3. Chen, T.: Highlights of statistical signal and array processing. *IEEE Signal Process. Mag.* **15**, 21–64 (1998). <https://doi.org/10.1109/79.708539>
4. Hua, C., et al.: Multipath map method for TDOA based indoor reverse positioning system with improved Chan-Taylor algorithm. *Sensors*. **20**, 1–14 (2020). <https://doi.org/10.3390/s20113223>
5. Zhang, Y., Gao, K., Zhu, J.: A TDOA-based three-dimensional positioning method for IoT. *Adv. Eng. Res.* **149**, 775–779 (2018). <https://doi.org/10.2991/mecae-18.2018.136>
6. Tay, S.I., Lee, T.C., Hamid, N.Z.A., Ahmad, A.N.A.: An overview of industry 4.0: definition, components, and government initiatives. *J. Adv. Res. Dyn. Control Syst.* **10**, 1379–1387 (2018)
7. Hong, J.M., Kim, S.H., Kim, K.J., Kim, C.G.: Multi-cell based UWB indoor positioning system. In: Nguyen, N.T., Gaol, F.L., Hong, T.-P., Trawiński, B. (eds.) ACIIDS 2019. LNCS (LNAI), vol. 11432, pp. 543–554. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14802-7_47
8. Lee, K., Hwang, W., Ryu, H., Choi, H.J.: New TDOA-based three-dimensional positioning method for 3GPP LTE system. *ETRI J.* **39**, 264–274 (2017). <https://doi.org/10.4218/etrij.17.0116.0554>
9. Kurkovsky, S., Williams, C.: Raspberry Pi as a platform for the internet of things projects: Experiences and lessons. In: Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, pp. 64–69 (2017). <https://doi.org/10.1145/3059009.3059028>
10. Carter, G.C.: Coherence and time delay estimation. *Proc. IEEE* **75**, 236–255 (1987). <https://doi.org/10.1109/PROC.1987.13723>
11. Gao, Y., Brennan, M.J., Joseph, P.F.: A comparison of time delay estimators for the detection of leak noise signals in plastic water distribution pipes. *J. Sound Vib.* **292**, 552–570 (2006). <https://doi.org/10.1016/j.jsv.2005.08.014>

12. Faerman, V., Voevodin, K., Avramchuk, V.: Frequency-domain generalized phase transform method in pipeline leaks locating. In: Communications in Computer and Information Science, pp. 22–38. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-23744-7_3
13. Glentis, G.O., Angelopoulos, K.: Leakage detection using leak noise correlation techniques - Overview and implementation aspects. In: PCI '19: 23rd Pan-Hellenic Conference on Informatics, pp. 50–57. ACM, New York (2019)
14. Spencer, S.J.: The two-dimensional source location problem for time differences of arrival at minimal element monitoring arrays. *J. Acoust. Soc. Am.* **121**, 3579–3594 (2007). <https://doi.org/10.1121/1.2734404>
15. Dalskov, D., Olesen, S.K.: Locating acoustic sources with multilateration applied to stationary and moving sources (2014). <http://www.es.aau.dk/sections/acoustics>. Accessed 10 Oct 2023
16. Qu, J., Shi, H., Qiao, N., Wu, C., Su, C., Razi, A.: New three-dimensional positioning algorithm through integrating TDOA and Newton's method. *EURASIP J. Wirel. Commun. Netw.* **2020**, 77 (2020). <https://doi.org/10.1186/s13638-020-01684-7>
17. Björklund, S.: A Survey and Comparison of Time-Delay Estimation Methods in Linear Systems. Linköpings universitet, Linköping (2003)
18. Brennan, M.J., Gao, Y., Joseph, P.F.: On the relationship between time and frequency domain methods in time delay estimation for leak detection in water distribution pipes. *J. Sound Vib.* **304**, 213–223 (2007). <https://doi.org/10.1016/j.jsv.2007.02.023>
19. Zhao, Z., Zi-Qiang, H.: The generalized phase spectrum method for time delay estimation. In: ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 46.2.1–46.2.4 (1984)
20. Faerman, V., Avramchuk, V., Voevodin, K., Sidorov, I., Kostyuchenko, E.: Study of generalized phase spectrum time delay estimation method for source positioning in small room acoustic environment. *Sensors*. **22**, 965 (2022). <https://doi.org/10.3390/s22030965>
21. Knapp, C.H., Carter, G.C.: The generalized correlation method for estimation of time delay. *IEEE Trans Acoust. ASSP* **24**, 320–327 (1976)
22. Fuchs, H.V., Riehle, R.: Ten years of experience with leak detection by acoustic signal analysis. *Appl. Acoust.* **33**, 1–19 (1991). [https://doi.org/10.1016/0003-682X\(91\)90062-J](https://doi.org/10.1016/0003-682X(91)90062-J)
23. Ma, Y., Gao, Y., Cui, X., Brennan, M.J., Almeida, F.C.L., Yang, J.: Adaptive phase transform method for pipeline leakage detection. *Sensors (Switzerland)* **19**, 310 (2019). <https://doi.org/10.3390/s19020310>
24. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for the FFT. In: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 1998, pp. 1381–1384 (1998). <https://doi.org/10.1109/ICASSP.1998.681704>
25. Rabiner, L.R., Schafer, R.W., Rader, C.M.: The Chirp z-transform algorithm. *IEEE Trans. Audio Electroacoust.* **AU 17**, 86–92 (1969). <https://doi.org/10.7551/mitpress/5222.003.0015>
26. Goertzel, G.: An algorithm for the evaluation of finite trigonometric series. *Am. Math. Mon.* **65**, 34–35 (1958)
27. Grado, L.L., Johnson, M.D., Netoff, T.I.: The sliding windowed infinite fourier transform: tips & tricks. *IEEE Signal Process. Mag.* **34**, 183–188 (2017). <https://doi.org/10.1109/MSP.2017.2718039>
28. Chicharo, J.F., Kilani, M.T.: A sliding Goertzel algorithm. *Signal Process.* **52**, 283–297 (1996)
29. Chauhan, A., Singh, K.M.: Recursive sliding DFT algorithms: a review. *Digit Signal Process.* **127**, 103560 (2022). <https://doi.org/10.1016/j.dsp.2022.103560>
30. Raspberry Foundation: Raspberry Pi 4 Model B (2021). <https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf>. Accessed 20 June 2023
31. Modul 9: Datasheet for HifiBerry DAC+ADC Pro (2020). <https://www.hifiberry.com/docs/data-sheets/datasheet-dac-adc-pro/>. Accessed 20 June 2023
32. Valgrind: Debugger and profiler valgrind-3.21.0. (2023). <https://valgrind.org/downloads/>. Accessed 10 Oct 2023

33. Faerman, V., Voevodin, K., Avramchuk, V.: Case of discrete-event simulation of the simple sensor node with CPN tools. In: International Siberian Conference on Control and Communications (SIBCON), pp. 1–9 (2022). <https://doi.org/10.1109/SIBCON56144.2022.10002956>
34. Jacobsen, E., Lyons, R., Communications, H., Associates, B., Lyons, R.G.: Sliding spectrum analysis. In: Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook, pp. 175–188. Wiley-IEEE Press (2012). <https://doi.org/10.1002/9781118316948.ch18>
35. Sysel, P., Rajmic, P.: Goertzel algorithm generalized to non-integer multiples of fundamental frequency. EURASIP J Adv Signal Process. **2012**(1), 56 (2012). <https://doi.org/10.1186/1687-6180-2012-56>
36. Sridharan, K., Babu, B.C., Kannan, P.M., Krithika, V.: Modelling of sliding goertzel DFT (SGDFT) based phase detection system for grid synchronization under distorted grid conditions. Procedia Technol. **21**, 430–437 (2015). <https://doi.org/10.1016/j.protcy.2015.10.065>