Intelligent Filtering Tuning in Edge Computing

Nieves G. Hernandez-Gonzalez, Juan Montiel-Caminos, Javier Sosa and Juan A. Montiel-Nelson
Institute for Applied Microelectronics (IUMA)
Department of Electronics Engineering and Automation (DIEA)
University of Las Palmas de Gran Canaria
Las Palmas de Gran Canaria, Spain
ORCID: 0000-0002-2401-4461, 0000-0001-5156-7646, 0000-0003-1838-3073, 0000-0003-4323-8097

2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS) | 979-8-3503-8717-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MWSCAS60917.2024.10658954

***Abstract*—An Artificial Intelligence approach to determine the best narrow input filter for a fundamental frequency extractor is presented. The algorithms are implemented on board a water current velocity sensor node. The target sensor node is based on an ARM Cortex-M0+ without DSP and FPU hardware support. The implementation is studied in detail in domains of real and integer variables. The results demonstrate that the proposed ANN-based solution is at least 5.5 and 15 times better than the published FFT solutions when using real variables and integer variables, respectively, for an input bandwidth reduction factor of 7.**

***Index Terms*—edge computing, artificial neural networks, sensor network**

I. Introduction

The extraction of parameters from acquired signals has traditionally been carried out by network servers, where there are no computational restrictions except purely economic ones. Edge computing techniques move those calculations, to the extent possible, from networked servers to the sensors themselves. The limitations of energy, computing resources and complexity of the calculation to be executed restrict the set of applications in which these techniques can be applied to current sensor nodes.

Regarding the extraction of parameters from the acquired signal, one of the most studied in the research literature is the identification of the fundamental frequency [1]. Most approaches in the literature mainly focus on demonstrating its quality in terms of accuracy. For this, the fast Fourier transform (FFT), Hilbert transform, finite impulse response filter (FIR), Kalman filtering, genetic algorithms and/or mathematical functions such as sine, square root and covariance are used [2]–[6]. The computational effort required for such approaches greatly exceeds the capabilities of current sensor nodes.

Furthermore, most fundamental frequency studies conclude that pre-filtering the raw signal improves results [2], [5]. In this sense, regardless of the methodology selected to extract the fundamental frequency, this work promotes a pre-filtering of the raw input data in the sensor node itself (edge-computing) before sending the raw data to the server for processing.

The paper is organized as follows. Section II presents the fundamental frequency extraction filtering problem. Also in this section the most relevant related literature is presented. Then, in Section III is introduced the proposed hardware-software architecture for intelligent filtering tuning. Next, in Section IV, its implementation and experiments are exposed in details. Finally, conclusions are presented in Section V.

II. Fundamental Frequency Extraction

Regardless of the final application, fundamental frequency extraction consists of obtaining the lowest frequency component contained in a given signal. Taking into account that according to Fourier all signals can be decomposed into a sum of sinusoidal signals, the frequency extraction algorithm assumes:

$$|\hat{u}(t)| = a_0 + \sum_{i=1}^{K} [a_i sen(2\pi i f t) + b_i cos(2\pi i f t)] \quad (1)$$

In particular, equation (1) models the velocity of a deep-water current. The coefficient a_0 represents the average value of the water flow movement. The sum of sinusoidal functions models the harmonic movement of water. In this equation, f is the so-called fundamental frequency. It is noteworthy that the fundamental frequency is the lowest frequency component present.

Fundamental frequency extraction applications are straightforward. Fig. 1 shows a generic application where the raw sensor data is filtered and then the extraction algorithm is applied. Nowadays it is very common for the sensor to have a digital output interface and incorporate a pre-filtering stage. With this, the computational requirements of the microcontroller that incorporates the sensor node are considerably reduced by executing this task efficiently on specific hardware designed for this purpose. Most of the filtering techniques implemented are based on averaging. Once, the averaging is performed, the acquisition subsystem provides the filtered data and the original raw data is lost. Finally, the objective of this filtering is to eliminate all data acquired outside the bandpass of interest is considered noise.

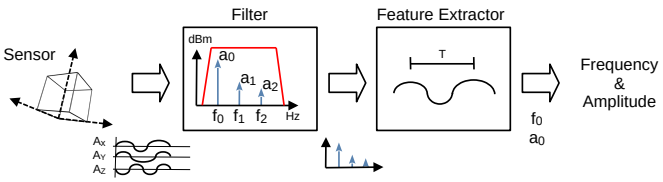


Fig. 1. Sensor node block diagram for fundamental frequency extraction.

However, since the extraction of the fundamental frequency is exclusively interested in the lowest frequency component and not in the rest of the components, the latter are considered noise. Therefore, its elimination before applying the extraction algorithm will facilitate its identification. However, its elimination is not an easy task, since the fundamental frequency varies in value.

A. Previous Works

Most recent and representative approaches in the literature on signal parameter extraction is presented in Table I. First at all, all approaches requires the usage of real numbers. Furthermore, published fundamental frequency extraction algorithms are expensive in terms of computational effort. Although all published research concludes that filtering the incoming signal improves the efficiency of the extraction procedure, only a couple of solutions preprocess the acquired data using a filter bank.

TABLE I
PUBLISHED SIGNAL PARAMETER EXTRACTION ALGORITHMS.

Ref.	Year	Functions	Lang.	Input Filter	Equipment
[2]	2019	Wavelet	Matlab	Bank	Desktop PC
[3]	2020	Kalman	Matlab	Fixed	FPGA, Cortex-M4
[4]	2021	FFT	C	Fixed	FPGA, Desktop PC
[5]	2022	Hilbert	Matlab	Bank	Desktop PC
[6]	2023	Kalman	Matlab	Fixed	Desktop PC

FFT: Fast Fourier Transform

The target application of the research presented in [5] is the estimation of the fundamental frequency of noisy speech. The authors define a set of temporal sequences and their associated frequencies. The solution them is based on the correlation between the set of sequences and the incoming signal divided in frames. Each temporal sequence included in the searching set is evaluated for each frame of acquired data. The correlated data then is normalized and compared to determine the fundamental frequency. Note that it is mandatory to use the Hilbert Transform to perform normalization and comparison during obtaining the fundamental frequency.

On the other hand, the authors in [2] introduce a classification algorithm applied to electromyogram (EMG) signals coming from a hand movement reaction. The solution is based on the Wavelet Transform and a filter bank. The approach requires at least 161 independent filters to fit the target application.

All solutions focus on demonstrating contribution in high-level languages such as Matlab. In addition to using advanced mathematical functions such as logarithmic, sine or Fourier, Hiltbert or Waveleft transform functions, parallelism techniques are widely used on them and, in particular, throughout the filtering.

Only the authors in [3] propose the use of a Kalman filter and an adaptive threshold to detect the fundamental frequency of the QRS complex from a human electrocardiogram (ECG) signal and the target implementation is an ARM Cortex-M4 microcontroller and/or an FPGA. Despite the low-level implementation in the microcontroller, the main advantage of this device is its own Floating Point Unit (FPU) running at 168 MHz.

As summary, all studied approaches in literature exploit of complex mathematical functions like Fourier, Hiltbert of Waveleft transforms that are highly complex to be implemented on the on-board ultra-low power microcontroller of a nowadays sensor node.

III. Intelligent Filtering Design

The hardware-software architecture proposed in this paper is presented in Fig. 2. Firstly, the data from the sensor is filtered using its own acquisition hardware. If this acceleration unit is not available (see Filter A in Fig. 2), this initial filtering is performed by software in the microcontroller. Then, the acquired data is filtered again (Filter B) and operated with the initially filtered data. In last stage, the frequency extraction algorithm determines the fundamental frequency and its amplitude.

The result of the performed subtraction operation is a band pass filter. Filter A with parameter n mainly defines the upper cutoff frequency and Filter B the lower cutoff frequency with parameter m. The characteristics of the band pass filter are determined by the prediction unit based on the evaluation of the initially filtered data.

The objective of the prediction unit is to determine the characteristics of Filter A and Filter B to adjust the center of the resulting band pass filter as closely as possible to the fundamental frequency to be extracted. In this sense, the proposed prediction unit evaluates the output of the Filter A and propose the n and m parameters.

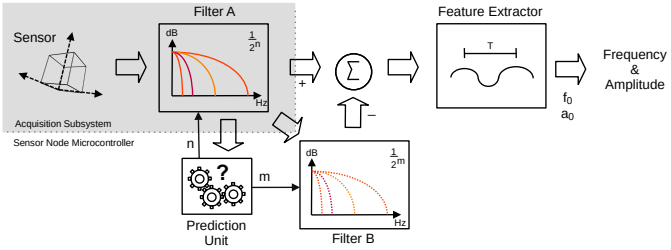


Fig. 2. Proposed filtering scheme for fundamental frequency extraction using ultra low power microcontroller.

A. Digital Filtering on the Edge

The most widely used filtering methodology is the finite impulse response (FIR) filter. As shown in (2), the output value of this filter, for a set of sampled input values, as the sum of those values weighted with different coefficients.

$$y(k) = \sum_{i=0}^{N-1} b_i x(k-i) \tag{2}$$

Despite its great usefulness, its computational requirements are considered high when it has to be implemented on microcontrollers without hardware support for real numbers.

However, by slightly reducing its performance as a filter, it can operate with powers of two coefficients so that it operates only on integers. In this sense, the weight b_i is formulated as $1/2^{c_i}$ and $c_i \in \mathbb{Z}$. Of Course, b_i belongs to rational numbers, but it is a power of two. Then, the weight multiplication function in Equation 2 can be implemented as a logical shift of the sampled value $x(k-i)$.

In addition, nowadays most acquisition subsystems allow filtering based on averaging. This is still a solution based on an FIR filter, where in (2) all weights b_i are set to $1/N$ where N is a power of two value. The attenuation rate is lower compared to other types of FIR filters, but its simplicity of implementation in hardware allows specific operating units to be included in the acquisition subsystems.

B. Prediction Unit

The proposed Prediction Unit (PU) is based on Artificial Intelligence (AI). Although there are multiple approaches in the literature that use these types of AI approaches, most of them are based on advanced computing infrastructures that use specific hardware accelerators, such as Compute Unified Device Architecture (CUDA) cores from Nvidia Corporation or high-end general purpose microprocessors. In the research literature, only a couple of solutions present implementations for microcontrollers such as the ARM Cortex-M4, with specific hardware including digital signal processor (DSP) instructions and a floating point unit (FPU).

In our approach, we use an artificial neural network (ANN). We choose this solution since it is the one that requires the least computational complexity from the point of view of the implementability of its mathematical functions. Fig. 3 shows the architecture of the proposed neural network to determine the the best band pass filter.

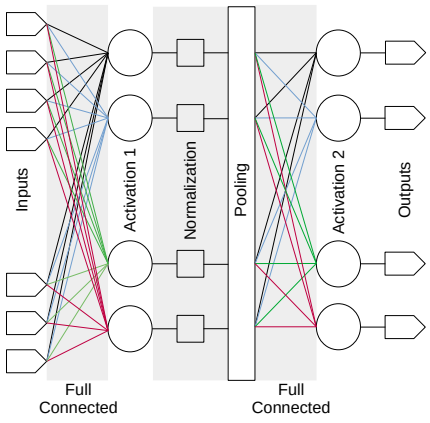


Fig. 3. Architecture of the proposed artificial neural network to be implemented in the ultra low power microcontroller.

Our proposal is composed of two activation layers, a single normalization layer and also a pooling layer following the sequence shown in Fig. 3. The activation layers and the pooling layer are fully connected with their previous stages.

On the other hand, it is well known that a neural network does not provide a general solution. Proof of this is that to obtain a solution based on this type of methodologies, it is necessary to carry out the training process with data from the specific application.

C. ANN Training

In this work we use the acquired raw data from [1]. Its target is to determine the fundamental frequency of water currents in an offshore oceanic aquaculture infrastructure. The sensing node/instrument is a water current meter based on the drag-tilt principle. From a purely electronic point of view, we used data acquired from a triaxial accelerometer (MMA8451Q, 3-axis, 14-bit digital accelerometer from NXP Corporation).

In order to obtain a training, testing and validation database, we operated the acquired data of accelerations using Matlab 2023 and its fast Fourier transform functionality to determine the involved fundamental frequencies. Once the database is ready, it is time to training the proposed ANN.

The sensor node sampling frequency is 12.5 samples per second and its acquisition subsystem memory is 32 samples length. In this scenario, the acquired data represents 2.56 seconds. Due to the limitations of the acquisition subsystem in terms of memory, the total number of samples can only be equal to or less than 32. And this is the reason for setting the number of proposed ANN inputs to 32. The total number of bandpass filters in the application is seven for the proposed filtering methodology (see Section III-A).

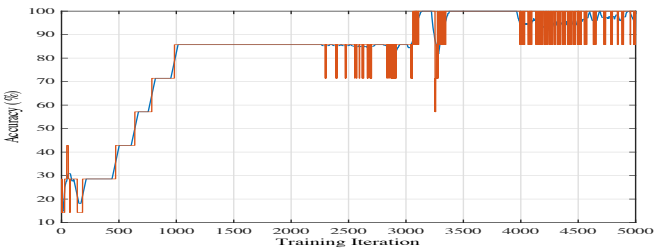
Therefore, the second activation layer, pooling layer, and output have seven elements. The number of elements in the first activation layer and the normalization layer are design variables. From a design point of view, the smaller the number of elements in those layers, the more optimized their implementation is. Based on the number of ANN inputs, the upper limit for this design variable is 32. We trained the ANN using from 32 to 3 elements in the first activation and the normalization stages. Since the available bandwidths are seven, we set to 7 the number of outputs. The number of elements of the pooling and second activation layers is also set to 7. In this way the ANN works as a classifier where only one output is active for every given set of input samples.

Fig. 4 presents the accuracy and loss during the training process of the previous described ANN. The training convergence is reached below the iteration 3.5k, where the loss obtained is close to 0.45. If we continue the training process it is possible to reduce the loss to a 0.25 when 5k iterations are executed.

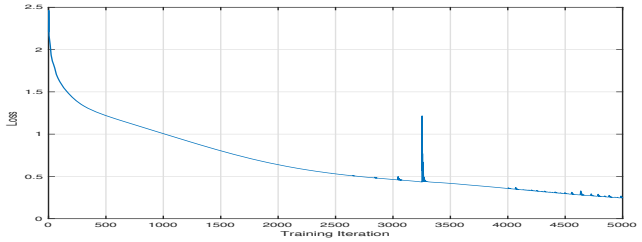
Once the training process is performed in Matlab, next step is to evaluate its implementation.

IV. Implementation and Experiments

As mentioned above, the target of the implementation is an ultra-low power microcontroller integrated with an ARM Cortex-M0+ in a water current velocity sensor node. The Cortex-M0+ does not include any DSP or FPU hardware support. The key question is what is the cost, in terms of computational effort, of implementing the high-level functionality required by ANNs, in particular their mathematical functions.



(a)



(b)

Fig. 4. ANN training process: a) Training convergence and b) losses.

A. Non-Integer Mathematical Functions

The lack of hardware support for real numbers makes it inevitable to use software libraries that provide such support. However, the data acquired is in the domain of integers. In this sense, most of the mathematical functionalities required by ANN can be performed in the domain of integers rather than in real numbers.

Table II presents the required machine cycles by the Cortex-M0+ to implement required mathematical functions for variables in float (real domain) and integer 32 bits. These values were obtained with GNU Arm Embedded Toolchain v10.3-2021.10. As summary, the compiler uses the same library functionalities to implement evaluated mathematical functions. Last column in this Table II presents the maximum difference between float and integer operation. This extra value is mainly due to the conversion functions from integer to float and vice versa.

TABLE II
COMPUTATIONAL EFFORT, MEASURED IN TERMS OF MACHINE CYCLES, OF THE MATHEMATICAL FUNCTIONS REQUIRED FOR THE ANN.

Math Function	**float**			**Int32**			**Max Diff**
	(min)	**(avg)**	**(max)**	**(min)**	**(avg)**	**(max)**	
e^x	7334	7493	7648	7399	7556	7713	+65
$\sqrt{x}$	6765	6946	7334	6797	6977	7359	+32
$sin(x)$	6372	6372	6374	6430	6442	6451	+77
$sinsh(x)$	4891	4891	4893	4960	4971	4988	+95
x/y	267	269	272	295	310	328	+56
$x * y$	179	180	183	10	10	10	-173

B. ANN

Once the ANN main parameters and weights are obtained during its design and training, it is time to its implementation. Table III presents the required number of cycles for each layer of the implemented ANN. Note that both the input and output layers require zero cycles. All input and output operations are performed through the microcontroller memory. The count of load and store operations implemented by the input and output layers are basically executed by the Full Connection 1 and Activation 2 layers respectively.

Although Full Connection layers have a different number of possible connections, we limit them to 3. That is the reason to require 562 or 82 cycles both Full Connection layers. The Normalization layer is based on function $\text{SoftMax}(x_i) = e^{x_i} / \sum_{j=0}^{N} e^{x_j}$. To speed up, the results in Table II, the integer version of the SoftMax function was implemented using a linear interpolation of the exponential function.

TABLE III
ANN: COMPUTATIONAL EFFORT REQUIRED FOR EACH STAGE.

Layer Name	**Size (neurons)**	**Real (float)**	**Integer (int32)**	**ratio**
Input	32	0	0	—
Full Connection	32	562	82	6.85
Activation 1	4	24	8	3.00
Normalization	4	23760	8984*1	2.64
Pooling	4	0	0	—
Full Connection	4	562	82	6.85
Activation 2	7	65	65	1.00
Output	7	0	0	—
Runtime*2(ms)	—	3.12	1.15	2.71

*1: e^x using linear interpolation ($|\text{MaxError}| < 1$ LSB).
*2: Assuming an 8MHz CPU frequency.

The ANN approach using real variables supported by software libraries takes only 3.12 ms in a ARM Cortex-M0+ at 8 MHZ. In addition, the advantage of the integer implementation is 2.71 times better than the double solution. Moreover, we implemented the FFT function included in the CMSIS-DSP library for ARM Cortex-M0+. In this experiment, we set the FFT length to 32 samples under the same running conditions. The FFT requires 17.31 ms, i.e., 5.55 times better for the real-domain ANN solution and 15 times better for the integer domain implementation.

A comparison in terms of accuracy with published works can be seen in Table IV. It should be noted that only references [3] and [4] could be implemented on ARM Cortex M0+, using floating-point library. The 92.40% accuracy of reference [5] corresponds to the experiments with a SNR of 5 dB, being these the best results of the reference [5]. In our work, the maximum accuracy is reached at iteration 3338 with a loss of 0.43, reducing the loss to 0.25 at iteration 5000.

V. CONCLUSIONS

In this work, we have presented a novel Artificial Intelligence approach to tune a programmable narrow band filter to the fundamental frequency of the incoming signal from a current wave velocity sensor node in real time. The target microcontroller is based on a ARM Cortex-M0+. The proposed ANN is studied in detail to be implemented without the usage of specific DSP or FPU hardward support. The comparison between proposed approach with an FFT-based

TABLE IV
COMPARISON WITH PUBLISHED SIGNAL PARAMETER EXTRACTION ALGORITHMS.

Reference	Target Language	Input Filter	Target Equipment	ARM Cortex M0+	Accuracy (%)
[2]	Matlab	Bank	Desktop PC	No	98.55
[3]	Matlab	Fixed	FPGA & Cortex-M4	Yes^{*1}	99.31
[4]	C	Fixed	FPGA & PC	Yes^{*1}	99.99
[5]	Matlab	Bank	Desktop PC	No	92.40
[6]	Matlab	Fixed	Desktop PC	No	NA
Ours	C	Bank	Cortex-M0+	Yes	99.85^{*2}

^{*1}: Using floating point library.
^{*2}: Iteration 3338 with a loss of 0.43, iteration 5000 with a loss of 0.25.

solution, we achieve a minimum improvement of 5.5 times with real domain variables, and 15 times better using integer variables.

ACKNOWLEDGMENT

This paper is part of the R+D+i projects PID2020-117251RB-C21, funded by MCIN/ AEI/ 10.13039/ 501100011033/, and TED2021-131470B-I00, funded by MCIN/ AEI/ 10.13039/ 501100011033/ and by the European Union NextGenerationEU/PRTR.

REFERENCES

[1] J. Montiel-Caminos, J. Sosa and J. A. Montiel-Nelson, "Tidal Current Fundamental Frequency Determination Algorithm for Integer Arithmetic Units," IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS'23), Tempe, AZ, USA, 2023, pp. 351-354

[2] Nishad, A.; Upadhyay, A.; Pachori, R.B.; Acharya, U.R. Automated classification of hand movements using tunable-Q wavelet transform based filter-bank with surface electromyogram signals. *Future Gener. Comput. Syst.* **2019**, *93*, 96–110.

[3] Zhang, Z.; Yu, Q.; Zhang, Q.; Ning, N.; Li, J. A Kalman filtering based adaptive threshold algorithm for QRS complex detection. *Biomed. Signal Process. Control* **2020**, *58*, 101827.

[4] Yifan, W.; Kai, C.; Xuan, G.; Renjun, H.; Wenjian, Z.; Sheng, Y.; Gen, Q. A High-Precision and Wideband Fundamental Frequency Measurement Method for Synchronous Sampling Used in the Power Analyzer. *Front. Energy Res.* **2021**, *9*, 652386.

[5] Queiroz, A.; Coelho, R. Noisy Speech Based Temporal Decomposition to Improve Fundamental Frequency Estimation. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2022**, *30*, 2504–2513.

[6] Jwo, D.-J.; Biswal, A. Implementation and Performance Analysis of Kalman Filters with Consistency Validation. *Mathematics* **2023**, *11*, 521.