Implementation of Embedded Multiple Signal Classification Algorithm for Mesh IoT Networks

Tiago Troccoli[†‡], Juho Pirskanen[†], Aleksandr Ometov[‡], Jari Nurmi[‡], Ville Kaseva[†]
[†] WIREPAS Ltd., Tampere, Finland
[‡] Faculty of Information Technology and Communication Sciences, Tampere University, Tampere, Finland
Contact email: tiago.troccolicunha@tuni.fi

***Abstract*—Angle-of-Arrival (AoA) methods are an Internet of Things (IoT) application, which could be used, for example, in indoor localization. Anchor nodes have an array of antennas and could send the data via Ethernet cable to the cloud that calculates AoA. However, having cable connections means high installation costs, and constantly transferring big chunks of data over some IoT networks, such as mesh, is energy inefficient. The solution of this paper consists in executing AoA locally in anchor nodes. Thus, the paper presents an implementation of a Multiple Signal Classification (MUSIC) algorithm tailor-made for embedded system devices. It calculates a complex eigendecomposition via an equivalent real formulation. It has a detailed memory analysis of the implemented solution that shows its memory requirements satisfy commercial embedded systems for IoT, such as Nordic semiconductor System-on-Chip (SoC) of nRF52 Series and all their SoCs with direction-finding capability. Experiments show that reducing the floating-point precision to shrink its memory footprint does not impact the accuracy. It also shows that minimizing the execution time of its time-consuming peak-finding operation has a few effects on accuracy.**

***Index Terms*—MUSIC, signal processing, angle-of-arrival, IoT, Mesh networks, embedded systems**

I. Introduction

Obtaining locations of different assets, such as tools, machines, or other equipment, is essential for efficient operations in many different industrial areas, and it is a common Internet of Things (IoT) application. For example, having the exact and frequently updated status of warehouse inventories can improve the efficiency of the operations significantly [1]. Even though Global Navigation Satellite System (GNSS) can provide accurate outdoor locations, using those solutions is not always practical. In indoor environments, satellite signals are affected by various propagation aspects, or providing additional positioning systems to the asset is not possible [2].

In network-based positioning, the locations of the assets are estimated in relation to the anchor nodes whose locations are known in advance [3]. Using Received Signal Strength Indicator (RSSI) to estimate distance is a well-known method, but its accuracy is typically limited to a few meters. Moreover, the accuracy is heavily dependent on the density of the anchors. Accuracy of a few meters can be acceptable in many use cases, but on some occasions need higher accuracy. For example, machine navigation for autonomous mobile robots, drones, industrial automation, or navigation systems that guide people in indoor environments. If such systems do not attain a good enough accuracy, industrial machines could be damaged, and people guided to wrong places. To obtain better accuracy, positioning based on Angle-of-Arrival (al (AoA) has been developed.

A typical architecture of the AoA solution in warehouses is composed of sensor nodes that are battery-powered embedded systems, and mains-powered anchor nodes with an antenna array connected with a high throughput data interface, i.e., Ethernet cable or fiber to cloud or server infrastructure. Anchor nodes perform reception of transmissions done by radio modules of assets (sensor nodes) and send the received in-phase and quadrature component (IQ) samples to the cloud where actual calculation of AoA and location are performed. However, having cable connections means high installation costs, and transferring IQ samples over some IoT networks, such as wireless mesh networks, is impractical. If that approach were used in such networks, anchor nodes would constantly send chunks of IQ values to a node so that the node would send them to another one, and so on, until reaching the gateway to transmit them to the cloud (See Figure 1). That would rapidly deplete the batteries of nodes, consume an unreasonable amount of radio, network, and computational resources, and increase the latency of AoA methods.

The solution in this research consists of executing AoA methods locally in anchor nodes instead of in the cloud. Thus, they only need to send the results of about $2-8$ bytes to the cloud. To carry out the solution, we implemented a Multiple Signal Classification (MUSIC) [4] in ANSI C99 tailor-made for embedded system devices equipped with a Uniform Linear Array of Antennas (ULA). The solution is intended to be executed in System-on-Chip (SoC) devices that have Bluetooth Low Energy (BLE) technology supporting Gaussian Frequency Shift Keying (GFSK) modulation. Although it could operate on other technologies, it was out of the scope of this research. As the anchor node would receive one packet transmission from a sensor node at a time, MUSIC estimates a single AoA during its execution.

The findings and contributions are as follow:

• A MUSIC algorithm tailor-made for embedded system devices that was successfully run in a Nordic Semiconductor System-on-Chip (SoC) with direction-finding capability. And $6,000$ tests were executed with different Signal-to-Noise Ratios (SNR), precision floating-points, and IQ values generated by clustered delay line E (CDL-E) channel model of MATLAB Communication Toolbox plus Additive White Gaussian

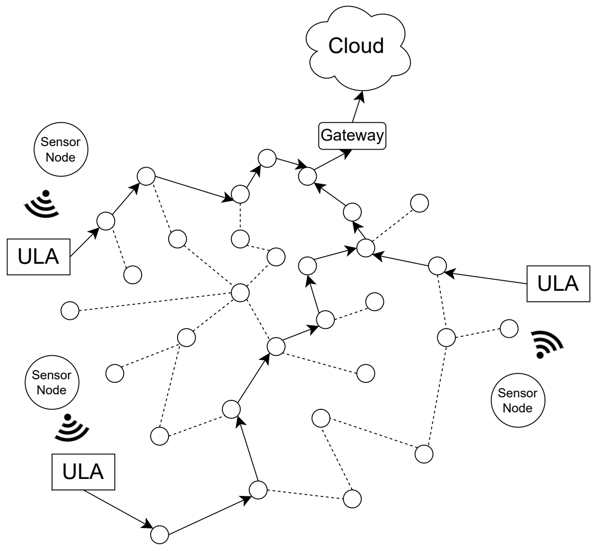

Fig. 1. Running AoA methods in the cloud is energy inefficient and expensive for Mesh IoT networks. The solution consists in executing them in anchor nodes.

Noise (AWGN).

• MUSIC satisfies the memory requirements for commercial embedded systems for IoT verified by this research. Such as even the most memory-constrained Nordic Semiconductor SoC of nRF52 Series and all their SoCs with direction-finding capability [5], [6] by the time this paper was written.

• The IQ matrix could take up more than 50% of overall RAM consumption. However, experiments show that reducing the floating-point precision to minimize memory consumption does not impact accuracy.

• Experiments show that finding the peak of the MUSIC spectrum is the most time-consuming operation by far. However, it is possible to lower its execution time in exchange for a few reductions in accuracy.

The paper is structured as follows. First, we provide the related works and mathematical background information in Section II and III. Afterward, the MUSIC implementation section has detailed explanations of under-the-hood algorithms alongside their intricacies in Section IV. Section V lays out the experiments and findings. The main results are summarized in the last section.

II. Related Works

In [7], [8], researchers developed unitary MUSIC on an FPGA. It is a version of MUSIC in which its complex covariance matrix is converted into a real one. In this paper, we did something similar but with a different approach. The implemented solution solves a complex eigendecomposition via an equivalent real formulation to be able to construct the complex noise subspace. Also, since their implementation was on FPGA, some operations were done in hardware.

In [9], the authors implemented MUSIC in a software-defined-radio and carried out real-world experiments. In [10], researchers proposed a smart antenna structure to execute AoA methods. In [11], MUSIC was programmed in a Digital Signal Processor (DSP) for AoA estimations for underwater acoustic sources. This paper differs from the others since it targets embedded systems for IoT with direction-findings capabilities.

There are many AoA methods, such as Estimation of Signal Parameters via Rotational Invariant Techniques (ESPRIT) [12], Space Alternating Generalized Expectation-Maximization (SAGE) [13], and Minimum Variance Distortionless Response (MDVR) [14]. In this research, we implemented MUSIC [4]. It is a very popular method and has been widely studied for decades resulting in multiple variations of the same algorithm. Moreover, many comparative studies reported that it attains great accuracy capability [15]–[17], and some researchers claim that it has superior performance compared to methods based on beamforming techniques [18], such as MDVR, which was one of the earliest categories of AoA methods. Although some researchers agree that algorithms based on the maximum likelihood approach have performance superior to MUSIC and ESPRIT, they are computationally very expensive, so they are unpopular [18], [19]. However, the downside of MUSIC consists in its time-consuming peak-finding operation (see eq. (6)), but it could be fairly decreased, as shown in this paper.

III. Mathematical Model Overview

MUSIC is classified as a subspace-based technique grounded on proprieties of the covariance matrix. The space spanned by the covariance's eigenvectors can be divided into two orthogonal subspaces, called signal and noise subspaces. Furthermore, the steering vectors correspond to the signal subspace, too. With those proprieties, it is possible to find the angle-of-arrival.

The MUSIC algorithm described in this paper is founded on a mathematical model that takes on the following premises [18]: the transmission medium is linear and isotropic, far-field assumption, and the sources generate narrowband signals propagated in an AWGN channel. Taking that into account, let's consider an ULA with M elements receiving signals generated by d sources. As shown in [20], the IQ sample for each source at a timestamp t is found by

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t), \tag{1}$$

where $\mathbf{s}(t) \in \mathbb{C}^{d\times 1}$ is a vector of signals of d sources, $\mathbf{n}(t) \in \mathbb{C}^{d\times 1}$ is a zero-mean spatially correlated additive noise and $\mathbf{A} \in \mathbb{C}^{M\times d}$ is the steering matrix, that is,

$$\mathbf{A} = \begin{bmatrix}\mathbf{a}(\theta_1) & \mathbf{a}(\theta_2) & ... & \mathbf{a}(\theta_d),\end{bmatrix} \tag{2}$$

where

$$\mathbf{a}(\theta_i)^T = \begin{bmatrix}1 & e^{j\mu_{\theta_i}} & e^{j2\mu_{\theta_i}} & ... & e^{j(M-1)\mu_{\theta_i}}\end{bmatrix}, \tag{3}$$

is the steering vector where $\mu_{\theta_i} = \frac{2\pi f_c}{c}\Delta \sin\theta_i$, c is the speed of light, f_c is the carrier frequency, and Δ is the distance between two adjacent antennas.

The algorithm is outlined as follows [18]:

1) Collect N samples $\mathbf{x}(t_n) \in \mathbb{C}^{M\times 1}$ for timestamp $t_1, t_2, ..., t_N$ and estimate the covariance matrix

$$\mathbf{R}_{xx} \approx \hat{\mathbf{R}}_{xx} = (\frac{1}{N})\mathbf{X}\mathbf{X}^H, \quad (4)$$

where $\mathbf{X} = \begin{bmatrix}\mathbf{x}(t_1) & \mathbf{x}(t_2) & ... & \mathbf{x}(t_N)\end{bmatrix} \in \mathbb{C}^{M\times N}$.

2) Compute the eigendecomposition of $\hat{\mathbf{R}}_{xx}$

$$\hat{\mathbf{R}}_{xx} = \mathbf{J}\mathbf{\Sigma}\mathbf{J}^H, \quad (5)$$

where $\mathbf{\Sigma}$ is a diagonal matrix comprised of eigenvalues and the matrix $\mathbf{J}$ is made up of eigenvectors.

3) Estimate the number of sources. Let d be this number.

4) Create a matrix $\mathbf{\Lambda}$ that consists of eigenvectors in $\mathbf{J}$ after removing the eigenvectors associated with the d greatest eigenvalues. The matrix $\mathbf{\Lambda}$ is the noise subspace of $\hat{\mathbf{R}}_{xx}$, which is orthogonal to the d steering vectors that make up $\mathbf{A}$.

5) Let's define $\mathbf{a}(\theta_i)$ as in eq. (3). Find the d largest peaks of the MUSIC spectrum

$$P(\theta_i) = \frac{1}{\mathbf{a}(\theta_i)^H\mathbf{\Lambda}\mathbf{\Lambda}^H\mathbf{a}(\theta_i)}, \quad (6)$$

where $-90° \leq \theta_i \leq 90°$. The angles corresponding to the d largest peaks are the angles of arrival.

The mathematical model does not consider correlated or coherent impinging signals, in the case of multiple signals sources. However, signals could attain such conditions in real-world environments, especially in rich multipath scenarios. Thus, to deal with this problem successfully, pre-processing techniques should be applied, such as Forward-backward averaging [21]. Also, the MUSIC applies ideal array steering vectors, which could degrade its accuracy. Therefore, array calibration could be used to lessen this problem [19]. Since this implementation only estimates one AoA at a time, we did not implement pre-processing techniques.

IV. MUSIC IMPLEMENTATION

This research implemented MUSIC from scratch using ANSI C99 programming language, considering embedded systems equipped with a uniform linear array of antennas, with GFSK modulation that is supported by short-range radio technologies, performing reception at a single operating channel. Thus, MUSIC can estimate only one AoA during its execution. Nevertheless, implementing MUSIC for embedded systems is not a simple task since it requires developing difficult numerical methods from the ground up. It uses three libraries from ANSI C: complex.h, math.h and stdint.h. Those libraries do not provide all numerical methods to execute MUSIC. Furthermore, well-known linear algebra libraries such as LAPACK [22] and Armadillo [23] are not designed for embedded systems. Although the Common Microcontroller Software Interface Standard (CMSIS) DSP Software Library is designed for them, it lacks some numerical methods used in MUSIC. For that reason and to make the implemented solution sufficiently compact, we implemented tailor-made algorithms. They include a modified version of complex matrix multiplication in which its result is a Hermitian matrix, the eigendecomposition called Jacobi method, a slight modification of the selection sort algorithm, the algorithm to construct the noise subspace, and the method to find the peak of MUSIC spectrum.

The IQ value matrix ($\mathbf{X}$) is the biggest matrix by far, and it consumes a considerable amount of RAM for an embedded system. For instance, if the number of antennas is 6 and samples is 250, it will consume 12000B or about 11.7KB in case the complex numbers are represented in single precision floating-point. Thus, storing its conjugate transpose, $\mathbf{X}^H$, to calculate the covariance matrix (see eq. (4)) would double this already big amount of memory consumption. The implemented solution does not store $\mathbf{X}^H$, since the matrix multiplication algorithm was modified. The standard way to multiply two matrices, $\hat{\mathbf{R}}_{xx} = \mathbf{X}\mathbf{Y}$, is

$$\hat{r}_{xx}(i,j) = (\frac{1}{N})\sum_{k=1}^{N} x(i,k) * y(k,j), \forall i,j = 1,...,M. \quad (7)$$

Since $\mathbf{Y} = \mathbf{X}^H$, $y(k,j) = \overline{x}(j,k)$, eq. (7) could be written as

$$\hat{r}_{xx}(i,j) = (\frac{1}{N})\sum_{k=1}^{N} x(i,k) * \overline{x}(j,k), \forall i,j = 1,...,M. \quad (8)$$

The implemented solution applies complex conjugate for each element whenever it is needed to avoid storing $\mathbf{X}^H$. Moreover, the implemented method only calculates the upper triangular elements of $\hat{\mathbf{R}}_{xx}$. For the lower part, it uses the Hermitian matrix property, that is, $\hat{r}_{xx}(i,j) = \overline{\hat{r}_{xx}}(j,i)$. To sum up, using these simple approaches, the device saves RAM in the order of MN and reduces the matrix computation by half.

The eigendecomposition is applied in the covariance matrix of $M \times M$, which is a small matrix since the number of antennas is small for a linear array of antennas in embedded system devices. The implemented solution uses the Jacobi eigenvalue algorithm, based on [24], to do the eigendecomposition in eq. (5). It is specifically devised for symmetric matrices, and it is considered a foolproof algorithm. Although it is not the most efficient method, it works fine for a small matrix [24], such as in this case, and it is simpler than the more sophisticated ones. However, the implemented Jacobi method works for real matrices. Thus, the solution converts the covariance matrix into a real one before calling it. In other words, it solves a complex eigendecomposition via an equivalent real formulation to be able to construct the complex noise subspace (matrix $\mathbf{\Lambda}$ from eq. (6)). Since the covariance is a complex (Hermitian) matrix, let's define it as $\hat{\mathbf{R}}_{xx} = \mathbf{A} + i\mathbf{B}$. Then the $M \times M$ complex eigendecomposition problem

$$(\mathbf{A} + i\mathbf{B}) \cdot (\mathbf{u} + i\mathbf{v}) = \lambda(\mathbf{u} + i\mathbf{v}) \quad (9)$$

can be formulated as $2M \times 2M$ real problem

$$\begin{bmatrix}\mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{u} \\ \mathbf{v}\end{bmatrix} = \lambda \begin{bmatrix}\mathbf{u} \\ \mathbf{v}\end{bmatrix}. \quad (10)$$

It is proved in [25] that the eigendecomposition (EVD) of the augmented problem of eq. (10) is

$$\begin{bmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{bmatrix} = \begin{bmatrix} \mathbf{U} & -\mathbf{V} \\ \mathbf{V} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma} \end{bmatrix} \begin{bmatrix} \mathbf{U} & -\mathbf{V} \\ \mathbf{V} & \mathbf{U} \end{bmatrix}^T, \quad (11)$$

where $\mathbf{U}+i\mathbf{V}$ or $-\mathbf{V}+i\mathbf{U}$ could be the matrix of eigenvectors of $\hat{\mathbf{R}}_{xx}$ with their respective matrix of eigenvalues, $\mathbf{\Sigma}$. The eigenvalues of $\hat{\mathbf{R}}_{xx}$ are found duplicated in the EVD of the augmented problem (11). For each duplicate valued pair of eigenvalues, (λ_i, λ_i), its respective eigenvectors are $\mathbf{u}_i + i\mathbf{v}_i$ and $-\mathbf{v}_i + i\mathbf{u}_i$. The eigenvectors of $\hat{\mathbf{R}}_{xx}$ can be one of them. As a result, the matrix of eigenvectors of $\hat{\mathbf{R}}_{xx}$ could be also a combination of $\mathbf{U} + i\mathbf{V}$ and $-\mathbf{V} + i\mathbf{U}$, provided that it has only one eigenvector of each duplicate valued pair of eigenvalues [24].

The real covariance matrix is 2 times greater than the complex one. Since the algorithm complexity of the Jacobi method is $O(N^3)$, the computational operation is in the order of 8 times greater. However, since the number of antennas in the ULA of embedded systems is sufficiently small, we can afford this modest additional operation.

However, the Jacobi method does not output ordered eigenvalues. Thus, after the EVD, the implemented solution sorts eigenvalues with their respective eigenvectors, so that the output looks like this $\lambda_1 \geq \lambda_1 \geq \lambda_2 \geq \lambda_2 \geq ... \geq \lambda_n \geq \lambda_n$. Therefore, the implemented method is able to pick out $\mathbf{u}+i\mathbf{v}$ or $\mathbf{v} - i\mathbf{u}$ for each duplicate valued pair of eigenvalues. However, the bottleneck of the sorting algorithm is the sorting of eigenvectors because it needs to swap all vector elements in every exchange of eigenvalues. Furthermore, the sorting algorithm does not need to be sophisticated since the number of eigenvalues is small. However, it should be good enough to overcome the bottleneck mentioned above. The implemented solution applies selection sort. It is a simple method that works fine for a small number of elements (eigenvalues), as in this case, with $O(n^2)$ number of comparisons [26]. However, it allows to concomitantly sort the eigenvectors with at most n swaps operation dealing successfully with the bottleneck.

A small code optimization was done to calculate eq. (6). Knowing that $a(\theta)^H \Lambda \Lambda^H a(\theta) = \|\Lambda^H a(\theta)\|$, the implemented solution sums the squared values of each elements of $\Lambda^H a(\theta)$, instead of computing three matrix multiplications. As a result, the device saves memory and time. Moreover, as previously mentioned, the algorithm searches for angles between $-90°$ and $90°$ that maximize eq. (6) by defining a step $0 < \Delta \leq 1$, so that the algorithm searches for angles $\theta_i = -90° + \Delta i$ for all $i = 0, 1, ..., \lfloor \frac{180°}{\Delta} \rfloor$. Since i is an integer number, to make sure the search includes $90°$, the Δ should be a number that divides $180°$ without a remainder.

V. Experiments and Findings

In the first part of the experiment, we checked the memory requirements for three floating-point types to run MUSIC and if they fit in commercial embedded systems with direction-finding capabilities. In the second part, we analyzed the impact of precision points on the MUSIC accuracy to reduce the memory consumption of the IQ value matrix ($\mathbf{X}$). We also carried out experiments to investigate the impact of the Δ parameter on the accuracy to reduce the function's execution time that finds the peak in the MUSIC spectrum since it is the most time-consuming function. In the third part, we evaluated the execution time of MUSIC on a real board using nRF52840. The number of antennas (M) was six with 50 samples (N) in all experiments. We verified some commercial linear array of antennas for embedded systems, and based on that, we considered six a reasonable number of antennas. Furthermore, according to our experiments, 50 samples are enough to get accurate estimations. The ULA is composed of isotropic antennas with frequencies 2 to 3 GHz. The ULA was developed using MATLAB Communication Toolbox as an intermediate step to generate IQ values that were fed to MUSIC executed in the embedded system.

Based on the single-precision floating-point MUSIC, we only changed the IQ values matrix ($\mathbf{X}$) to the half-precision one, since only $\mathbf{X}$ has a considerable impact on RAM. However, to turn $\mathbf{X}$ matrix into double-precision, we changed the entire code to double-precision as well to avoid losing precision points obtained by $\mathbf{X}$. Moreover, the *complex.h* library of C does not have support for a half-precision floating-point, so we created a C structure made up of real and imaginary parts shown below.

```
/* The complex library of C does not have support
   for half-precision floating point, so we created
    a C structure shown below.*/
struct COMPLEX_NUM{
    __fp16 real;
    __fp16 imag;
};
typedef struct COMPLEX_NUM Complex;
```

Table I shows memory requirements for three floating-point types. To generate those consumptions, we used a bare-metal cross-compiler that comes with GNU ARM Embedded Toolchain. We considered an nRF52833 SoC to configure the compilation since it has a direction-finding capability and an ARM Cortex-M4 with a Floating-Point Unit (FPU). The half floating point is under IEEE 754-2008 format. The hardware floating-point instructions and hardware floating-point linkage were enabled, that is, the *-mfloat-abi=hard* flag was set. However, those memory consumption generated by *arm-none-eabi-size.exe* command line do not take into account the memory consumption inside functions (stack and heap memory). It provides the size of *bss*, *text* and *data* memory sections that made up RAM and ROM. Thus, to get the full picture, we generated the stack memory consumption using the flag *-fstack-usage* for the single-precision floating-point MUSIC (Table II) only. It should be noted that the implemented solution does not consume heap memory.

Thus, we can conclude that this MUSIC implementation satisfies the memory requirements for commercial embedded systems for IoT verified by us, such as Nordic Semiconductor System-on-Chip (SoC) nRF52 Series and all their SoCs with direction-finding capability [5], [6] by the time this paper was written. Also, Table II shows the relative execution time for

TABLE I
MEMORY CONSUMPTION FOR EACH FLOATING POINT
($N = 50$ AND $M = 6$).

	RAM consumption	ROM consumption
Half precision floating-point	2.58KB	13.87KB
Single precision floating point	3.97KB	12.03KB
Double precision floating point	10.98KB	19.16KB

TABLE II
STACK MEMORY AND RELATIVE EXECUTION TIME VALUES FOR SINGLE PRECISION FLOATING POINT ($\Delta = 0.1$, $N = 50$ AND $M = 6$)

Function	Stack memory consumption	Relative execution time (%)
covmat_conversion	24B	0.02%
get_covmat	48B	0.62%
music_function	64B	-
find_peak	40B	95.02%
evdcmp	184B	4.25%
eigen_sort	32B	-
get_noise_matrix	40B	0.07%
runMUSIC	16B	-

each function considering $\Delta = 0.1$, $N = 50$ and $M = 6$. Considering a method with n functions, we define a relative execution time of a function f_i as

$$r_i = \frac{t_i}{\sum_{k=1}^{n} t_k}, \tag{12}$$

where t_i and r_i are, respectively, the absolute and relative execution time of the function f_i for $1 \leq i \leq n$. In other words, the relative execution time of a function measures the fraction or the percentage of its absolute execution time relative to the execution time of the method. Some functions have a dash. In the case of function runMUSIC, it is the main one and calls other functions. The eigen_sort is called by get_noise_matrix, so the running time is in get_noise_matrix's execution time. The same reason for music_function, which is called by find_peak. It is clear that find_peak is the most time-consuming function. Therefore, we conducted experiments to reduce its relative execution time by means of increasing the Δ parameter to verify its impact on accuracy.

Notably in Table I, there is a substantial increment in ROM consumption for double-precision floating-point, because the FPU of ARM-Cortex M4 does not support that precision, but only single-precision. Thus, C runtime library functions execute all the calculations in software instead of in hardware, i.e., the calculations are emulated [27], [28]. As a result, more ROM is required to execute operations using double precision. However, the ARM-Cortex M4 processor carries out calculations in half-precision using single-precision instructions [29]. To do that, it promotes half-precision numbers into single-precision ones before executing arithmetic operations. That is, both precisions use the same instructions for such operations, thus, their ROM usage is similar. The small difference between them is probably due to the data structure mentioned previously. Due to executing mathematical operations, the half-precision solution needs to access the numbers in such a data structure that requires more instructions, while the difference in RAM usage is a clear consequence of the distinct number of bits among different floating point precisions.

Table III shows the results of Mean Squared Errors (MSE) in degrees between the difference of 100 random angle of arrivals and their estimations by MUSIC for each pair of $SNR \times \Delta$ considering three distinct precision floating-point formats. To do that, the MUSIC was fed by IQ values generated by the CDL-E channel model from MATLAB Communications Toolbox [30] plus AWGN. The experiments were conducted in a Raspberry Pi 3 Model B since it has an ARM processor with half-precision floating-point capability. Note that Table III shows SNR of 0 dB. Although this low SNR could be impractical for wireless systems, the experiments only considered the accuracy of MUSIC method without taking into account whether system operation at such a low SNR value is viable.

The main takeaway of Table III is that the change of precision in floating-point does not impact the accuracy. We can also easily see that the accuracy improves with the increase of SNR as expected. Moreover, the rise of the Δ parameter generally deteriorates the accuracy by a small amount. However, very small fluctuations are verified between 0.25 and 0.75. Since the accuracy variations are small between Δ values, increasing Δ has a small impact on the accuracy, but a fairly substantial effect on the relative execution time of find_peak function as shown in Figure 2.

TABLE III
MSE IN DEGREES FOR EACH PAIR OF SNR $\times$ Δ FOR EVERY PRECISION FLOATING-POINT.

Half-precision floating point					
Delta (Δ) / SNR [dB]	0.1	0.25	0.5	0.75	1.0
0	3.11	3.09	3.09	3.10	3.17
10	1.92	1.94	1.95	1.99	2.00
20	1.75	1.74	1.83	1.73	1.81
30	1.72	1.73	1.78	1.75	1.80
Single-precision floating point					
Delta (Δ) / SNR [dB]	0.1	0.25	0.5	0.75	1.0
0	3.11	3.09	3.09	3.10	3.17
10	1.94	1.94	1.95	1.95	2.00
20	1.75	1.74	1.83	1.99	1.81
30	1.72	1.73	1.78	1.73	1.80
Double-precision floating point					
Delta (Δ) / SNR [dB]	0.1	0.25	0.5	0.75	1.0
0	3.11	3.09	3.09	3.10	3.17
10	1.94	1.94	1.95	1.99	2.00
20	1.75	1.74	1.83	1.73	1.81
30	1.72	1.74	1.78	1.75	1.80

Figure 3 shows the execution time in milliseconds of MUSIC using a single-precision floating-point for each value of Δ keeping constant IQ values generated by the CDL-

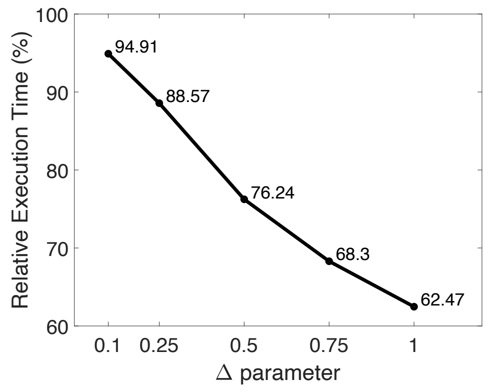

Fig. 2. Relative execution time of find_peak method for each value of Δ. The find_peak is the most time-consuming function, so, increasing its Δ parameter has a significant impact on MUSIC execution time.

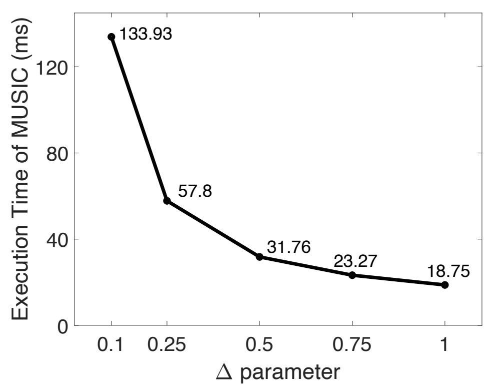

Fig. 3. Execution time of MUSIC using single-precision floating-point for each value of Δ.

E channel model from MATLAB Communications Toolbox plus AWGN. It was run in a PCA10056 development board that has an nRF52840 equipped with an ARM Cortex-M4 with FPU as shown in Figure 4. To measure the time, we used Saleae Logic Analyzer. The PCA10056 executed MUSIC periodically. Every time before the execution, it turned on a pin (rising edge), and after the completion, it turned off the same pin (falling edge). The time difference between the rising and falling edge is the execution time of MUSIC. From Figure 3, when the Δ increases from 0.1 to 0.25, the execution time shrinks abruptly by more than a half. That is a significant improvement considering that the accuracy is almost the same (see Table III). However, the execution time of find_peak function still consumes most part of the MUSIC's execution time by a great amount, which is 88.57% as it is shown in Figure 2. Furthermore, as the relative execution time of find_peak function decreases, the execution time of MUSIC declines as well, with initial values falling sharply and final ones reducing smoothly. Therefore, it is clear that the impact of find_peak function on execution time becomes less significant as the Δ increases. To conclude, increasing the Δ parameter can speed up MUSIC significantly in exchange for compromising its accuracy by a small amount.

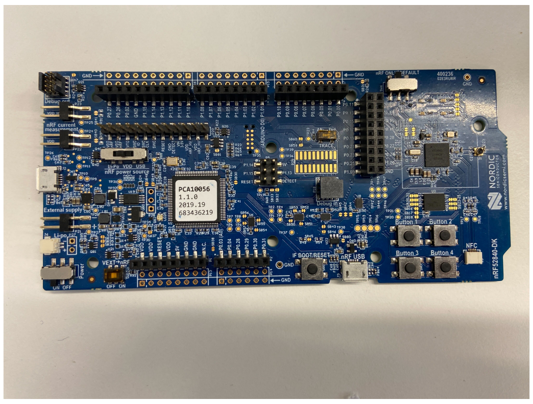

Fig. 4. The PCA10056 development kit ran MUSIC to measure its execution time.

VI. Conclusions

This paper presented a MUSIC implementation for commercial embedded systems equipped with a ULA that uses Bluetooth Low Energy technology. By implementing MUSIC and obtaining AoA estimation at embedded devices, the communication requirements are drastically reduced compared to a situation where IQ samples would be transferred to the cloud. These savings directly impact radio resources, system energy consumption, and the cost of mesh IoT networks. Further, it is foreseen to enable a significant increase in the total number of indoor localization events that a single system could produce, which can be especially important, e.g., in warehouses.

Notably, we identified the possibility to mitigate the IQ matrix footprint by decreasing its floating-point precision without degrading the accuracy. Furthermore, it is possible to reduce the execution time of the most time-consuming method (find_peak) in exchange for a few reductions in accuracy. Moreover, memory requirements satisfy commercial SoC with BLE direction-finding capabilities verified in this research, such as nRF5340 and nRF52833. MUSIC implementation is completed. However, we plan to study further state-of-the-art code optimization methods and consider other AoA estimation algorithms and communication systems aspects. Finally, we expect to conduct real-world experiments with a set of antenna arrays and test deployments.

Acknowledgment

The authors gratefully acknowledge funding from European Union's Horizon 2020 Research and Innovation programme under the Marie Skłodowska Curie Grant Agreement No. 956090 (APROPOS, http://www.apropos-itn.eu/).

References

[1] Y. Lu, M. Gerasimenko, R. Kovalchukov, M. Stusek, J. Urama, J. Hosek, M. Valkama, and E. S. Lohan, “Feasibility of location-aware handover for autonomous vehicles in industrial multi-radio environments,” *Sensors*, vol. 20, no. 21, p. 6290, 2020.

[2] J. Torres-Sospedra, I. Silva, L. Klus, D. Quezada-Gaibor, A. Crivello, P. Barsocchi, C. Pendão, E. S. Lohan, J. Nurmi, and A. Moreira, “Towards ubiquitous indoor positioning: Comparing systems across heterogeneous datasets,” in *2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2021, pp. 1–8.

[3] P. Pascacio, S. Casteleyn, J. Torres-Sospedra, E. S. Lohan, and J. Nurmi, “Collaborative indoor positioning systems: A systematic review,” *Sensors*, vol. 21, no. 3, p. 1002, 2021.

[4] R. Schmidt, “Multiple emitter location and signal parameter estimation,” *IEEE transactions on antennas and propagation*, vol. 34, no. 3, pp. 276–280, 1986.

[5] Nordic Semiconductor, “Bluetooth direction finding,” https://www.nordicsemi.com/Products/Bluetooth-Direction-Finding, 2022.

[6] Nordic Semiconductors, “nrf52 series,” https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52.html, 2022.

[7] M. Kim, K. Ichige, and H. Arai, “Implementation of FPGA Based Fast DOA Estimator Using Unitary MUSIC Algorithm [Cellular Wireless Base Station Applications],” in *Proc. of IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, vol. 1. IEEE, 2003, pp. 213–217.

[8] M. Kim, K. Ichige, and H. Ari, “Real-time smart antenna system incorporating FPGA-based fast DOA estimator,” in *Proc. of 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, vol. 1. IEEE, 2004, pp. 160–164.

[9] M.-C. Hua, C.-H. Hsu, and H.-C. Liu, “Implementation of direction-of-arrival estimator on software defined radio platform,” in *Proc. of 8th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)*. IEEE, 2012, pp. 1–4.

[10] H. Wang and M. Glesner, “Hardware implementation of smart antenna systems,” *Advances in Radio Science*, vol. 4, no. C. 2, pp. 185–188, 2006.

[11] S.-Y. Hou, S.-H. Chang, H.-S. Hung, and J.-Y. Chen, “DSP-based implementation of a real-time DOA estimator for underwater acoustic sources,” *Journal of Marine Science and Technology*, vol. 17, no. 4, pp. 320–325, 2009.

[12] R. Roy and T. Kailath, “Esprit-estimation of signal parameters via rotational invariance techniques,” *IEEE Transactions on acoustics, speech, and signal processing*, vol. 37, no. 7, pp. 984–995, 1989.

[13] J. A. Fessler and A. O. Hero, “Space-alternating generalized expectation-maximization algorithm,” *IEEE Transactions on signal processing*, vol. 42, no. 10, pp. 2664–2677, 1994.

[14] S. A. Vorobyov, “Principles of minimum variance robust adaptive beamforming design,” *Signal Processing*, vol. 93, no. 12, pp. 3264–3277, 2013.

[15] N. Dheringe and B. Bansode, “Performance evaluation and analysis of direction of arrival estimation using MUSIC, TLS ESPRIT and Pro ESPRIT algorithms,” *Perform. Eval*, vol. 4, no. 6, pp. 4948–4958, 2015.

[16] R. Feng, Y. Liu, J. Huang, J. Sun, and C.-X. Wang, “Comparison of music, unitary esprit, and sage algorithms for estimating 3d angles in wireless channels,” in *2017 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 2017, pp. 1–6.

[17] O. A. Oumar, M. F. Siyau, and T. P. Sattar, “Comparison between music and esprit direction of arrival estimation algorithms for wireless communication systems,” in *The First International Conference on Future Generation Communication Technologies*. IEEE, 2012, pp. 99–103.

[18] Z. Chen, G. Gokeda, and Y. Yu, *Introduction to Direction-of-arrival Estimation*. Artech House, 2010.

[19] E. Tuncer and B. Friedlander, *Classical and modern direction-of-arrival estimation*. Academic Press, 2009.

[20] P.-J. Chung, M. Viberg, and J. Yu, “Doa estimation methods and algorithms,” in *Academic Press Library in Signal Processing*. Elsevier, 2014, vol. 3, pp. 599–650.

[21] S. U. Pillai and B. H. Kwon, “Forward/backward spatial smoothing techniques for coherent signal identification,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 1, pp. 8–15, 1989.

[22] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney *et al.*, *LAPACK Users' guide*. SIAM, 1999.

[23] C. Sanderson and R. Curtin, “Armadillo: a template-based C++ library for linear algebra,” *Journal of Open Source Software*, vol. 1, no. 2, p. 26, 2016.

[24] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[25] D. Day and M. A. Heroux, “Solving complex-valued linear systems via equivalent real formulations,” *SIAM Journal on Scientific Computing*, vol. 23, no. 2, pp. 480–498, 2001.

[26] R. Sedgewick, *Algorithms in C++, parts 1-4: fundamentals, data structure, sorting, searching*. Pearson Education, 1998.

[27] J. Yiu, *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors*. Newnes, 2013.

[28] I. Johnson. (2022) 10 useful tips for using the floating point unit on the cortex-m4. [Online]. Available: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/10-useful-tips-to-using-the-floating-point-unit-on-the-arm-cortex--m4-processor

[29] Arm, *Arm Compiler armclang Reference Guide Version 6.12*. Arm Ltd., 2019.

[30] MATLAB. (2022) Communications toolbox: Design and simulate the physical layer of communications systems. [Online]. Available: https://se.mathworks.com/products/communications.html