Fast Real-World Implementation of a Direction of Arrival Method for Constrained Embedded IoT Devices

Tiago Troccoli*
tiago.troccoli@wirepas.com
WIREPAS Ltd., Faculty of Information Technology and Communication Sciences, Tampere University
Tampere, Finland

Juho Pirskanen
WIREPAS Ltd.
Tampere, Finland

Aleksandr Ometov
Faculty of Information Technology and Communication Sciences, Tampere University
Tampere, Finland

Jari Nurmi
Faculty of Information Technology and Communication Sciences, Tampere University
Tampere, Finland

Ville Kaseva
WIREPAS Ltd.
Tampere, Finland

ABSTRACT

Direction of arrival (DOA) methods are found in many applications, and in the case of the Internet of Things (IoT), it is used for indoor localization. However, the implementation of DOA in IoT devices poses a real challenge, since they are computationally expensive complex numerical methods that could easily lead to resource starvation, unacceptable execution time, and rapid depletion of batteries of small constrained embedded systems typically found in IoT networks. This paper contributes to alleviating that problem, it presents a fast low-power optimized version of a DOA method called Unitary TLS ESPRIT. The optimization exploits the radio communication system design to avoid two time-consuming executions of eigendecomposition, and instead, it applies two simple Power Method algorithms. The result is a lightweight version of ESPRIT that can attain sub-millisecond execution time. To prove the solution's viability, we carried out experiments on energy consumption, memory footprint, accuracy, and execution time for three floating-point formats in a commercial constrained embedded IoT device series without any operating system and software layers. Experiments show the solution satisfies the hardware requirements and the floating-point precision fully operated by the Floating-Point Unit is found to be the best option.

CCS CONCEPTS

• **Hardware** → **Digital signal processing**; • **Computer systems organization** → *Firmware*; *Embedded software*.

KEYWORDS

direction of arrival, signal processing, embedded systems, ESPRIT

1 INTRODUCTION

Direction of arrival (DOA) methods are found in many applications, such as medical appliances, radar, navigation, military devices, and indoor localization systems in the case of the Internet of Things (IoT) [8]. It has gained popularity in the IoT industry since 2019 with the advent of Bluetooth Low Energy (BLE) Direction Finding technology that unlocked the capability of estimating DOA from BLE devices possibly achieving centimeter-level location accuracy [23, 32]. That is a significant achievement since RSSI-based BLE solutions typically have an accuracy of a few meters [2]. IoT networks are composed of many nodes that are typically low-cost battery-powered constrained embedded devices. For DOA purposes, some of them are equipped with an array of antennas (anchor nodes) that receive signals from tags. It is possible to estimate DOA from those signals, thereafter, computing the location of tags. Due to the intrinsic complexity of DOA methods, a reasonable approach consists in executing them in the cloud. That is impractical in some network topologies such as mesh IoT networks [27].

To be specific, anchor nodes would need to constantly transfer big chunks of data (signals) from one node to another until reaching the destination (cloud), rapidly depleting their batteries. The size of such chunks depends on the number of samples and antennas, but it can easily exceed more than one kilobyte. Another possibility involves deploying internet cables to anchor nodes, however, that would require an increment in expenses. A low-cost and low-power solution consists of executing a DOA method in anchor nodes as depicted in Figure 1. So, nodes would transfer only few bytes (depending on the floating-point precision) instead of kilobytes. However, the implementation of DOA algorithms in IoT devices poses a real challenge, since such devices are typically battery-powered constrained embedded systems with very limited computational resources, in contrast, DOA methods are composed of resource-hungry and time-consuming complex numerical algorithms that could easily lead to rapid depletion of batteries, unacceptable execution time and resource starvation. On top of that, IoT devices usually execute some small tasks concurrently, such as collecting data from sensors to compute physical phenomena and communicating with other devices. In such a multi-threaded scenario, running DOA methods could be even more difficult, and management of computational resources needs to be thought carefully.

Papers about real-world implementations of DOA methods are quite uncommon. In [26] and [25], researchers implemented single-source MUSIC and TLS ESPRIT in LabView NI hardware using a real uniform linear array of antennas. There were some implementations of different versions of MUSIC and ESPRIT in FPGA [3, 16, 33] as well. A development of MUSIC based on parallel computing

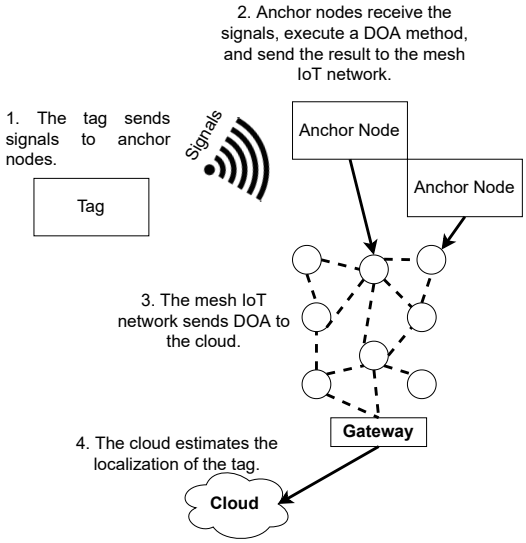


Figure 1: Overview of low-cost solution for mesh IoT networks.

was successfully devised for a Digital Signal Processor (DSP) [17]. Additionally, an implementation of two-source MUSIC for Software Defined Radio platforms was accomplished [12] as well as a one-source MUSIC for constrained embedded devices [27].

To alleviate the cited computational burden, this research takes a novel approach and propose an optimized implementation of a DOA method called ESPRIT tailor-made for commercial constrained IoT devices written in the C99 programming language without requiring any external libraries besides ones from C99 Standard Library. The implemented solution was tested with 6000 DOAs generated by a Clustered Delay Line E (CDL-E) channel model with Additive white Gaussian noise (AWGN).

However, the implemented solution estimates a single DOA. DOA methods such as ESPRIT can estimate multiple DOAs during their execution, so radar applications sending sounding signals and measuring when their own signal is received from different reflections can take full advantage of that capability by identifying multiple copies of their own reflected signal. However, in IoT radio communication systems where anchors are employed to locate multiple tags, this is not possible in practice with low-cost single receiver anchor nodes operating at a single RF channel at a given time such as in Bluetooth receivers [19]. That is, if more than one tag sends a signal to an anchor node, at the same time and frequency resources, the signal to interference and noise ratio would be too low for that radio receiver to detect transmission reliably. For example, a receiver could not be able to decode both transmitter IDs of the transmitters reliably as each transmission would interfere with the other. Therefore, in this scenario, DOA methods can only estimate a single DOA only.

Seeing that the implemented solution is composed of numerical methods, it is imperative to find a suitable floating-point precision. Ideally, the precision should be as small as possible to attain a minimum memory footprint while maintaining acceptable accuracy, and the lowest energy consumption with the fastest execution time. Unfortunately, that does not match the reality. Thus, the aim of the experiments consisted in investigating empirically the impact of different floating-point precision on accuracy, memory usage, execution time, and energy consumption. And additionally, to check the viability of the solution on commercial constrained embedded systems. The experiments intended to find answers to the following questions:

(1) Does the implemented solution satisfy the memory requirements for commercial constrained IoT devices?
(2) Is the solution battery operable for low battery capacity?
(3) Does the decrease of floating-point precision deteriorate accuracy? For how much? Does that deterioration (if any) make up for the presumed lower execution time, energy, and memory consumption?
(4) Conversely, does the increase of floating-point precision improve accuracy? For how much? Does that improvement (if any) make up for the presumed higher execution time, energy, and memory consumption?
(5) What is the most time-consuming function?

2 STANDARD ESPRIT

This paper does not go over all the intricate mathematical details that prove how ESPRIT works under the hood, but rather it gives an algorithmic level overview of the implemented solution of the research. Reference [5] provides detailed mathematical analysis of ESPRIT algorithm. Let's consider an uniform linear array of antennas (ULA) with M elements receiving d signals from far-field sources i.e. tags impinging the array at angles $\theta_1, \theta_2, \ldots, \theta_d$. Let's assume they are narrowband signals propagated in an AWGN channel with linear and isotropic transmission medium. In the mathematical model [6], the IQ sample for each source at a timestamp t is

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

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,

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

where

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

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.

ESPRIT is a popular method that was devised after MUSIC. It takes advantage of the shift-invariance propriety of subarrays in the ULA in such a way that it eliminates the search for peaks in the spectrum which is a time-consuming operation, doing so, it could be faster than MUSIC. The ESPRIT method divides the ULA into two subarrays. In the implemented solution, the two subarrays are composed of $m = M - 1$ consecutive antennas, and $M - 2$ overlapping ones as shown in the Figure 2. It is possible to create those subarrays by multiplying the steering matrix $\mathbf{A}$ by

$$\begin{aligned}\mathbf{J}_1 &= [\mathbf{I}_m \quad \mathbf{0}_m] \in \mathbb{R}^{m\times M},\\ \mathbf{J}_2 &= [\mathbf{0}_m \quad \mathbf{I}_m] \in \mathbb{R}^{m\times M},\end{aligned} \tag{4}$$

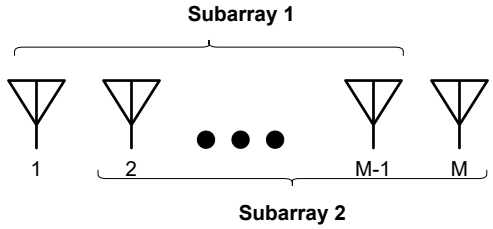


Figure 2: The implemented ESPRIT divides the ULA into two subarray of size $m = M - 1$ with $M - 2$ overlapping antennas.

where $\mathbf{I}_m$ is the identity matrix and $\mathbf{0}_m$ is a column vector of zeroes, both with size m. It can be shown that the shift-invariance propriety of the d steering vectors $(\mathbf{a}(\theta_i))$ is expressed as $\mathbf{J_1 A \Phi} = \mathbf{J_2 A}$, in which $\mathbf{\Phi} = \text{diag}[e^{j\mu_{\theta_1}}, e^{j\mu_{\theta_2}}, \ldots, e^{j\mu_{\theta_d}}] \in \mathbb{C}^{d\times d}$. However, instead of computing $\mathbf{\Phi}$, ESPRIT takes another approach and it estimates the subspace rotational operation $\mathbf{\Psi}$ from the signal subspace $(\mathbf{U}_s)$ as shown in

$$\mathbf{J_1 U}_s \mathbf{\Psi} \approx \mathbf{J_2 U}_s, \tag{5}$$

which also has information about DOAs. Note that the equality does not hold in the shift-invariance equation 5. Because the signal subspace is estimated from the covariance matrix which is also an estimation. As a result, equation 5 may not have an exact solution, therefore, ESPRIT estimates $\mathbf{\Psi}$ via least squares (LS) or total least squares (TLS). The Algorithm 1 gives an overview of how the Standard ESPRIT works.

Algorithm 1: Standard ESPRIT

Input: The IQ values matrix $\mathbf{X}$.
Output: The estimated DOAs: $\hat{\theta}_1, \ldots, \hat{\theta}_d$.

1. Collect N IQ samples $\mathbf{x}(t_n) \in \mathbb{C}^{M\times 1}$ for timestamp $t_1, t_2, \ldots, t_N$ and estimate the covariance matrix
$$\mathbf{R}_{xx} \approx \hat{\mathbf{R}}_{xx} = (\frac{1}{N})\mathbf{X}\mathbf{X}^H,$$
where $\mathbf{X} = \begin{bmatrix}\mathbf{x}(t_1) & \mathbf{x}(t_2) & \ldots & \mathbf{x}(t_N)\end{bmatrix} \in \mathbb{C}^{M\times N}$.
2. Apply the eigendecomposition (EVD) in $\hat{\mathbf{R}}_{xx}$ to find the signal subspace $\mathbf{U}_s$.
3. Create the matrices $\mathbf{J_1}$ and $\mathbf{J_2}$ as defined in equation 4 and solve the shift invariance equation below by applying LS or TLS to estimate $\mathbf{\Psi}$.
$$\mathbf{J_1 U}_s \mathbf{\Psi} \approx \mathbf{J_2 U}_s.$$
4. The eigenvalues of $\mathbf{\Psi}$ contain the DOAs. So, compute the eigenvalues of $\mathbf{\Psi}$ which is $\mathbf{\Phi} = \text{diag}[\phi_1, \phi_2, \ldots, \phi_d]$ to estimate the DOAs
$$\mu_i = \arg(\phi_i), \quad \hat{\theta}_i = \arcsin(-\frac{\lambda\mu_i}{2\pi\Delta}), \quad 1 \le i \le d.$$

3 UNITARY TLS ESPRIT WITH ROW WEIGHTING

The implemented solution of this research is the Unitary TLS ESPRIT with row weighting in C99 programming language tailor-made for embedded systems which is a variation of the Standard ESPRIT. TLS stands for Total Least Squares [18], Unitary means that all matrices are converted into real ones by applying a unitary transformation for Centro-Hermitian matrices, and row weighting is an operation that is a mathematically proven way to improve the accuracy of ESPRIT [29]. The advantage of this variation over the standard ESPRIT is to avoid complex matrices that require more memory footprint and computations than real ones. Also, in general, numerical methods are usually designed taking into account real numbers instead of complex values. Although those methods may also work for complex matrices, they could require more computations which translates into more execution time. Furthermore, TLS has a better estimation than LS [5, 22, 28] which consequently improves the accuracy which is defined as the difference between the estimated angle and the actual angle.

Let $\mathbf{\Pi}_p \in \mathbb{C}^{p\times p}$ be any anti-diagonal identity matrix, that is,

$$\mathbf{\Pi}_p = \begin{bmatrix} 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & \ldots & 1 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 1 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \end{bmatrix},$$

and $\mathbf{Q}_n \in \mathbb{C}^{n\times n}$ be an unitary transform matrix defined as

$$\mathbf{Q}_{2n} = \frac{1}{\sqrt{2}} = \begin{bmatrix} \mathbf{I}_n & j\mathbf{I}_n \\ \mathbf{\Pi}_n & -j\mathbf{\Pi}_n \end{bmatrix} \quad \text{or}$$

$$\mathbf{Q}_{2n+1} = \frac{1}{\sqrt{2}} = \begin{bmatrix} \mathbf{I}_n & 0 & j\mathbf{I}_n \\ \mathbf{0}_n^T & \sqrt{2} & \mathbf{0}_n^T \\ \mathbf{\Pi}_n & 0 & -j\mathbf{\Pi}_n \end{bmatrix},$$

depending if its size is even or odd. The algorithm is outlined below.

(1) Collect N IQ samples $\mathbf{x}(t_n) \in \mathbb{C}^{M\times 1}$ for timestamp $t_1, t_2, \ldots, t_N$ and estimate the covariance matrix
$$\mathbf{R}_{xx} \approx \hat{\mathbf{R}}_{xx} = (\frac{1}{N})\mathbf{X}\mathbf{X}^H, \tag{6}$$
where $\mathbf{X} = \begin{bmatrix}\mathbf{x}(t_1) & \mathbf{x}(t_2) & \ldots & \mathbf{x}(t_N)\end{bmatrix} \in \mathbb{C}^{M\times N}$. Convert the complex covariance matrix into a real one, that is, $\mathbf{C} = \text{Re}\{\mathbf{Q}_M^H\, \hat{\mathbf{R}}_{xx}\, \mathbf{Q}_M\} \in \mathbb{R}^{M\times M}$.
(2) Apply eigendecomposition (EVD) to find the signal subspace $\mathbf{U}_s$ of the real covariance matrix $\mathbf{C}$. The signal subspace is composed of eigenvectors corresponding to the d largest eigenvalues.
(3) Let's define $\tilde{\mathbf{J}}_2 = [\mathbf{0}_m \quad \mathbf{W}] \in \mathbb{R}^{m\times M}$. The matrix $\mathbf{W} = \text{diag}\left[1, \sqrt{2}, \sqrt{3}, \ldots, \sqrt{w}, \sqrt{w}, \sqrt{w}, \ldots, \sqrt{3}, \sqrt{2}, 1\right]$ are composed of the weights, in which $w = \min\{m_s, M - m_s + 1\}$ determines where the increase in weighting stops and $m_s \in \mathbb{R}_{>0}$ is defined such that $m_s \le M$ [29].
(4) Thereafter, let's define $\mathbf{K_1} \triangleq 2\,\text{Re}\{\mathbf{Q}_m^H\, \tilde{\mathbf{J}}_2\, \mathbf{Q}_M\}$ and $\mathbf{K_2} \triangleq 2\,\text{Im}\{\mathbf{Q}_m^H\, \tilde{\mathbf{J}}_2\, \mathbf{Q}_M\}$ as unitary transformation of $\tilde{\mathbf{J}}_1$ and $\tilde{\mathbf{J}}_2$, respectively.
(5) Estimate the matrix $\mathbf{Y} \in \mathbb{R}^{d\times d}$ from the equation below
$$\mathbf{K_1 U}_s \mathbf{Y} \approx \mathbf{K_2 U}_s, \tag{7}$$

by means of TLS. To do that, first compute the matrix

$$\mathbf{E} = \begin{bmatrix} (\mathbf{K_1 U}_s)^T \\ (\mathbf{K_2 U}_s)^T \end{bmatrix} \begin{bmatrix} (\mathbf{K_1 U}_s) & (\mathbf{K_2 U}_s) \end{bmatrix}.$$

Since $\mathbf{E} \in \mathbb{R}^{2d \times 2d}$ is a real symmetric matrix, then it is diagonalizable [13], thus we can apply EVD resulting in $\mathbf{E} = \mathbf{V}\Sigma\mathbf{V}^T$. In which $\Sigma = \text{diag}[\sigma_1, \sigma_2, \ldots, \sigma_{2d}]$ is the matrix of its eigenvalues in such a way that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{2d}$. Let's partition these two matrices into four ones

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} \\ \mathbf{V}_{21} & \mathbf{V}_{22} \end{bmatrix} \qquad \text{and} \qquad \Sigma = \begin{bmatrix} \Sigma_1 & \mathbf{0} \\ \mathbf{0} & \Sigma_2 \end{bmatrix},$$

thus the right sub-matrix $\begin{bmatrix} \mathbf{V}_{12} & \mathbf{V}_{22} \end{bmatrix}^T$ are made up of eigenvectors associated with the d smallest eigenvalues. If $\mathbf{V}_{22}$ is non-singular, then $\mathbf{Y} = -\mathbf{V}_{12}\mathbf{V}_{22}^{-1}$. Otherwise, there is no solution for the TLS [18].

(6) The eigenvalues of $\mathbf{Y}$ contain information about direction of arrivals. So, the next step is to compute them which are $\mathbf{\Omega} = \text{diag}[\omega_1, \omega_2, \ldots, \omega_d]$ and therefore extract the DOAs by the following operations

$$\mu_i = 2\arctan(\omega_i), \qquad \theta_i = \arcsin(-\frac{\lambda\mu_i}{2\pi\Delta}), \qquad 1 \leq i \leq d, \tag{8}$$

where λ is the wavelength.

4 OPTIMIZED ESPRIT

The implemented solution optimizes Unitary TLS ESPRIT by exploiting the radio communication design where only a single tag transmits a signal at a time as discussed in Section 1. It means that the signal subspace ($\mathbf{U}_s$) and the right sub-matrix ($\begin{bmatrix} \mathbf{V}_{12} & \mathbf{V}_{22} \end{bmatrix}^T$) of step 2 and 5 of Section 3 respectively, become vectors. As a result, the implemented solution can avoid calculating the time-consuming eigendecompositions (EVD), and instead, it applies the simple Power Method and Inverse Power Method. In doing so, the execution time of ESPRIT is drastically reduced, since these two methods compute only the eigenvector that is of interest from the solution point of view while EVD computes all eigenvectors and eigenvalues. And computing all of them requires a very complicated and time-consuming algorithm. Notably, the complexity of the QR Algorithm, a typical method for EVD, is $6n^3 + O(n^2)$ per iteration [31], not to mention the Hessenberg decomposition that must be done before the QR Algorithm, and an algorithm of $O(n)$ to find the eigenvector with the highest or lowest eigenvalue. While the Power Method and Inverse Power Method have a complexity of $O(n^2)$ and $O(n^3)$ per iteration respectively, they already calculate the desired eigenvector, require very simple computations, and experimentally we found they converge mostly in 4 iterations only in our solution.

The objective of the optimization is to reduce the memory consumption and execution time of ESPRIT to attain satisfactory portability to run in constrained embedded systems. Thus, all the algorithms were implemented from scratch in C99 programming language, except the inverse of sine, the inverse of tangent, and squared root which are functions from *math.h*. The tailor-made numerical methods include Power Method, Inverse Power Method, Total Least Squares, and small under-the-hood algorithms. Since one of the objectives of the implemented solution is to attain a minimal memory footprint as much as possible, it does not use *complex.h* library from C programming language. Instead, it has a data structure for complex numbers with two variables representing the real and imaginary parts, and functions for complex multiplication, addition, and conjugation. Notably, the implemented ESPRIT only employs *math.h* and *stdint.h* libraries reassuring its minimal computational resources consumption goal and portability.

The first and simpler optimization concerns the equation 6. As discussed in (hidden reference), the matrix $\mathbf{X}$ is big since it contains the IQ samples that are complex numbers. If $N = 200$ and $M = 6$, the matrix $\mathbf{X}$ would occupy 9.375KB of RAM, considering single-precision floating-point, that would be a big memory consumption for constrained embedded devices. By estimating the covariance matrix (eq. 6), the code may have to store temporary matrix $\mathbf{X}^H$ as well, which would double the RAM usage. The implemented solution does not store $\mathbf{X}^H$. The standard way to multiply two matrices, $\hat{\mathbf{R}}_{xx} = \mathbf{XY}$, is

$$\hat{r}_{xx}(i,j) = (\frac{1}{N}) \sum_{k=1}^{N} x(i,k) * y(k,j), \forall i, j = 1, \ldots, M. \tag{9}$$

Since $\mathbf{Y} = \mathbf{X}^H$, then $y(k,j) = \overline{x}(j,k)$, therefore eq.(9) could be written as

$$\hat{r}_{xx}(i,j) = (\frac{1}{N}) \sum_{k=1}^{N} x(i,k) * \overline{x}(j,k), \forall i, j = 1, \ldots, M. \tag{10}$$

Moreover, since $\hat{\mathbf{R}}_{xx}$ is Hermitian, which means $\hat{r}_{xx}(j,i) = \overline{\hat{r}_{xx}}(i,j)$, thus the solution applies matrix multiplication only on its upper triangular part, therefore the equation 10 becomes

$$\begin{aligned} \hat{r}_{xx}(i,j) &= (\frac{1}{N}) \sum_{k=1}^{N} x(i,k) * \overline{x}(j,k), \\ \hat{r}_{xx}(j,i) &= \overline{\hat{r}_{xx}}(i,j), \\ &\forall i, j = i, \ldots, M. \end{aligned} \tag{11}$$

From equation 11, the implemented solution estimates the covariance matrix using the same matrix $\mathbf{X}$ twice by applying element-wise conjugate transpose operation, thus it does not need to store $\mathbf{X}^H$. Furthermore, it only computes the upper triangular part of $\hat{\mathbf{R}}_{xx}$. To sum up, that approach saves execution time by half and RAM usage in the order of MN. That is an important improvement, since calculating the covariance matrix is the most time-consuming function, as shown in the next section.

Since there is only one tag transmitting at a time ($d = 1$), it is unnecessary to apply the complicated and time-consuming EVD in step 2 of Section 3 to get all eigenvectors and eigenvalues, as the signal subspace ($\mathbf{U}_s$) contains only one eigenvector. Instead of EVD, the solution applies Power Method. It is a simple algorithm that only computes the eigenvalue with the greatest absolute value [11] and its corresponding eigenvector, which is the signal subspace as demonstrated in next paragraph. By doing so, the implemented solution saves execution time and memory footprint.

To Power Method guarantee to converge, the matrix must be diagonalizable, there must exist only one eigenvalue with the greatest absolute value and it must be a real number [9]. For example, considering $\lambda_i \in \mathbb{R}, i = 1, \ldots, M$ to be eigenvalues of a diagonalizable matrix, if $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_M|$ then the cited matrix satisfies

the convergence requirements. The matrix $\mathbf{C}$ is a real covariance matrix, thus it is symmetric [24], therefore it is diagonalizable and its all eigenvalues are real numbers [10]. Moreover since $\mathbf{C}$ is a real covariance matrix, it is positive semi-definite [24], which means all eigenvalues are non-negative. The line-of-sight (LOS) component of the received signal that constitutes the eigenvalue of the signal subspace is greater than eigenvalues of the noise subspace [4], and since they are all non-negative, it is not possible to have eigenvalues of the noise subspace equal to or greater than one of signal subspace in magnitude. Therefore, the eigenvalue of the signal subspace is the greatest in magnitude.

The implemented Power Method (Algorithm 2) does not compute the eigenvalue since the solution only needs the eigenvector. We considered $K = 30$ and $tol = 10^{-6}$. We carried out thousands of experiments and we verified that in most cases the Algorithm 2 takes 4 to 5 iterations to converge, and 30 iterations are much more than enough in all experimental instances, hence for $k > 30$ we assume the algorithm fails to compute the signal subspace.

Algorithm 2: Power Method

Input: covariance matrix $\mathbf{C}$.
Output: signal subspace $\mathbf{U}_s$.
Define $\mathbf{v}_1 = [1, 1, \ldots, 1]^T \in \mathbb{R}^M$, $\mathbf{v}_0 = \mathbf{0}_M$, $k = 1$, $tol \ll 1 \in \mathbb{R}_{>0}$, and $K \in \mathbb{Z}_{>0}$.
while $k \leq K$ ***and*** $\|\mathbf{v}_k - \mathbf{v}_{k-1}\|^2 > tol$ **do**
  $\mathbf{v}_{k+1} \leftarrow \mathbf{C}\mathbf{v}_k$
  $\mathbf{v}_{k+1} \leftarrow \frac{\mathbf{v}_{k+1}}{\|\mathbf{v}_{k+1}\|}$
  $k \leftarrow k + 1$
end
if $k > K$ **then**
  /* Convergence failed */
  return *NULL*
end
else
  /* Convergence succeeded */
  return $\mathbf{v}_k$
end

In TLS, the matrix $\mathbf{E}$ has a size of 2×2, since $d = 1$. As a result, the right sub-matrix of its eigenvector matrix ($\mathbf{V}$), that is, $\begin{bmatrix}\mathbf{V}_{12} & \mathbf{V}_{22}\end{bmatrix}^T$ is a vector of size 2. Since these elements are scalars, we redefine the sub-matrix to $\begin{bmatrix}v_{12} & v_{22}\end{bmatrix}^T$. Similarly, to the previous optimization, instead of applying the EVD which is a complicated numerical method, the solution applies the Inverse Power Method. This method only computes the smallest eigenvalue in magnitude and its corresponding eigenvector [9], which is the vector $\begin{bmatrix}v_{12} & v_{22}\end{bmatrix}^T$. To the Inverse Power Method works, matrix $\mathbf{E}$ must be non-singular and it must have only one smallest real eigenvalue in modulus. For example, $|\lambda_1| > |\lambda_2|$, for $d = 1$ and $\lambda_1, \lambda_2 \in \mathbb{R}$. Since $\mathbf{E}$ is a real covariance matrix, therefore it is a real symmetric matrix having real eigenvalues, more specifically it is positive semi-definite. It could be positive definite if the vectors $\mathbf{K_1U}_s$ and $\mathbf{K_2U}_s$ are linear independent [24], in that case, the matrix $\mathbf{E}$ is non-singular. However, if $\mathbf{K_1U}_s \approx \mathbf{K_2U}_s$, the matrix $\mathbf{E}$ could be "almost singular" (ill-conditioned). To make sure it is always non-singular, a method should turn it into a positive definite matrix. Taking advantage of its positive semi-definiteness characteristic, a well-known simple method consists in a small perturbation, that is, $\mathbf{E} \approx \mathbf{E} + \alpha\mathbf{I}_{2d}$ [15].

In which α should be a small positive number to make the perturbed matrix close to the non-perturbed one. Assuming matrix $\mathbf{E}$, after the perturbation, to have only one smallest eigenvalue, it is possible to apply the Inverse Power Method, Algorithm 3. The implemented algorithm only computes the eigenvector since its eigenvalue is not used. Moreover, we defined $\alpha = 10^{-5}$, $tol = 10^{-6}$ and $K = 30$. Again, these values were found empirically. Furthermore, the implemented Inverse Power Method does not apply LU decomposition to find the solution of linear equations repeatedly as recommend. Consider the linear system $\mathbf{E}\mathbf{v}_{k+1} = \mathbf{v}_k$ to have the elements shown in equation 12, since $\mathbf{E}$ has a size of 2×2, we can easily find the solution of linear equations analytically as

$$\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} = \begin{bmatrix}a\\ b\end{bmatrix}, \tag{12}$$

$$\begin{cases} x_1 = \dfrac{a_{22}a - a_{12}b}{a_{11}a_{22} - a_{12}a_{21}} \\ x_2 = \dfrac{a_{11}b - a_{21}a}{a_{11}a_{22} - a_{12}a_{21}} \end{cases}. \tag{13}$$

Algorithm 3: Inverse Power Method

Input: covariance matrix $\mathbf{E}$.
Output: vector $\begin{bmatrix}v_{12} & v_{22}\end{bmatrix}^T \in \mathbb{R}^2$.
Define $\mathbf{v}_1 = [1, 1, \ldots, 1]^T \in \mathbb{R}^M$, $\mathbf{v}_0 = \mathbf{0}_M$, $k = 1$, $tol \ll 1 \in \mathbb{R}_{>0}$, $\alpha \in \mathbb{R}_{>0}$, and $K \in \mathbb{Z}_{>0}$.
/* Apply a small perturbation */
$\mathbf{E} = \mathbf{E} + \alpha\mathbf{I}_{2d}$
while $k \leq K$ ***and*** $\|\mathbf{v}_k - \mathbf{v}_{k-1}\|^2 > tol$ **do**
  /* Solve $\mathbf{E}\mathbf{v}_{k+1} = \mathbf{v}_k$ (equation 13) */
  $\mathbf{v}_{k+1} \leftarrow solve(\mathbf{E}, \mathbf{v}_k)$
  $\mathbf{v}_{k+1} \leftarrow \frac{\mathbf{v}_{k+1}}{\|\mathbf{v}_{k+1}\|}$
  $k \leftarrow k + 1$
end
if $k > K$ **then**
  /* Convergence failed */
  return *NULL*
end
else
  /* Convergence succeeded */
  return $\mathbf{v}_k$
end

Notably, it is also possible to apply the Power Method in $\mathbf{E}$ instead of the Inverse Power Method. Since $\mathbf{E}$ is a 2×2 real symmetric matrix with distinct eigenvalue assumption, its two eigenvectors are orthogonal to each other [10]. Thus, applying the Power Method in $\mathbf{E}$, results in a vector $\mathbf{v} \in \mathbb{R}^2$. Any vector orthogonal to $\mathbf{v}$ can be the eigenvector corresponding to the smallest eigenvalue. However, the execution time, accuracy, and memory consumption of the ESPRIT using this small variation are almost the same as one with the Inverse Power Method. So, to make this paper concise enough we did not go over it. Additionally, in step 5 of section 3, since $d = 1$

the matrix $\mathbf{Y}$ is scalar. So, it can be computed as follow $\mathbf{Y} = -v_{12}/v_{22}$, and its eigenvalues calculation is not applicable.

5 EXPERIMENTS

The experiments considered three floating-point formats under IEEE 754-2008 specification, so there were three versions of the implemented solution all with $N = 50$ (number of IQ samples), $M = 6$ (number of antennas) and $m_s = 4$ (weighting stop number). These numbers are highly dependent on hardware and indoor environments, so there is no rule of thump. However, 6 antennas are small enough for embedded systems operating in IoT networks and 50 samples fit in the constrained memory of such devices. Moreover, we found that this configuration provided sufficiently good accuracy in our simulation considering the CDL-E channel model with AGWN. Additionally, the three floating-point formats are abbreviated as follows:

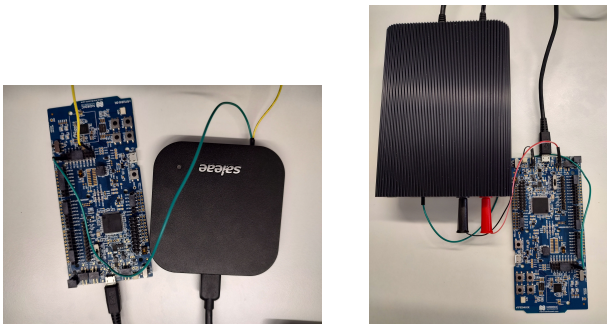
- FP16: half-precision floating-point.
- FP32: single-precision floating-point.
- FP64: double-precision floating-point.

The first version of the solution used FP32. The second version employed FP16 to reduce the big memory consumption of the IQ matrix ($\mathbf{X}$) only, the other matrices remained in FP32 since there are small. The third version operated completely in FP64. We chose those formats since Arm Cortex-M4 processors can operate in FP16 and FP32, and the C compiler of Arm processors can compute or emulate FP64.

5.1 Experimental setup

Firstly, all the measurements consider $N = 50$, $M = 6$ and $m_s = 4$ as explained previously. To measure the memory footprint (RAM and ROM), execution time, and energy consumption as shown in Table 2, we employed a PCA10056 development kit that comes with an nRF52840 System-on-Chip (SoC) having an Arm Cortex-M4 of 64MHz with Floating-Point Unit (FPU). No operating system and software layers were used. The hardware floating-point instructions and hardware floating-point linkage (*-mfloat-abi=hard*) were activated. All devices of the nRF52 series have support for BLE, although nRF52840 does not have Bluetooth Direction Finding capability, it is almost identical to other nRF52 and nRF53 devices that do have it. Additionally, we measured the stack memory consumption of main functions for FP32 as shown in Table 3. There is no dynamic memory usage. Furthermore, we measured the relative execution time for primary functions as presented in Table 3. The relative execution time is defined as the running time of a function relative to the total execution time of the implemented method in percentage. Note that the inverse_power_method has a '-', due to its execution time is included in the TLS function since the latter calls the former. We did not consider the running time of runESPRIT as well, since it calls all other functions, so its execution time is the same as the implemented solution.

Notably, to measure the energy consumption, we employed a power measurement tool as pictured in Figure 3b. However, for debugging reasons we needed to activate a general-purpose input/output (GPIO) port. Thus, the energy usage is slightly overestimated. To measure the execution time, we utilized a logic analyzer as shown in Figure 3a. In both scenarios, the implemented solution ran in an infinity loop, a GPIO was set high and low before and after the execution of the algorithm. So, we could check when the method started and finished to properly carry out the two measurements.



(a) (b)

Figure 3: Figure (a) shows a logic analyzer (right) that was used to measure the execution of the solution run in a PCA10056 (left). Figure (b) shows a power measurement tool (left) that was used to measure the energy of the solution run in a PCA10056 (right).

Moreover, we calculated the method's Mean Squared Error (MSE) of the accuracy considering 500 DOAs for each floating-point precision × SNR pair as shown in Table 1. In total, we analyzed 6000 different DOAs. In mathematical terms, the MSE of the accuracy is

$$MSE = \frac{1}{L}\sum_{i=1}^{L}(\alpha_i - \hat{\alpha}_i)^2, \tag{14}$$

where $L = 500$ is the number of estimated DOAs, α_i and $\hat{\alpha}_i$ is actual DOA and its estimation in degrees, respectively. We generated the IQ values using the CDL-E channel model with AGWN provided by MATLAB Communication Toolbox. The defined ULA was composed of isotropic antennas with frequencies 2 to 3 GHz. The execution time, energy consumption, and memory footprint were done in a PCA1005. However, if we used the PCA1005 in MSE of the DOA's accuracy, we would have to execute manually 6000 executions. So, to automate the MSE of the DOA's accuracy we employed a Raspberry Pi Model B. It has also an Arm processor capable of operating in FP16, FP32, and FP64.

	MSE (in degrees)			
Precision \ SNR [dB]	**0**	**10**	**20**	**30**
FP16	1.9210	1.0787	1.0042	0.9983
FP32	1.9205	1.0788	1.0049	0.9982
FP64	1.9205	1.0788	1.0049	0.9982

Table 1: The MSE of the accuracy (eq. 14) of 500 DOAs for each precision × SNR pair ($N = 50$ and $M = 6$).

5.2 Results and Discussion

Before answering the questions raised in Section 1, a short explanation should be highlighted to understand the values of Table 2. The FPU of Arm Cortex-M4 does have support for FP32 only. The FP16

	RAM Usage	ROM Usage	Execution Time	Energy Consumption
FP16	3.12kB	8.33kB	1.318ms	6.92nWh
FP32	4.90kB	9.80kB	0.855ms	4.77nWh
FP64	8.68kB	18.17kB	18.317ms	97.5nWh

Table 2: Memory footprint, execution time and energy consumption of each floating-point format ($N = 50$ and $M = 6$)

is used as a storage format only. When operating in FP16, the processor promotes FP16 into FP32 before and demotes it after every calculation [1]. Those operations create a small overhead that could increase the ROM consumption and the execution time. A slower execution time may translate into more energy consumption. As a result, we can see that the two previously mentioned measurements of FP16 are higher than those of FP32. Moreover, the FPU of Arm Cortex-M4 does not have support for FP64 at all. Thus, the C compiler emulates FP64 calculations [14, 34], that emulation creates an excess of computations culminating in a substantial increment of execution time and ROM usage. In fact, the execution time of FP64 is about 20 times slower than FP32.

Considering the memory usages shown in Table 2 and 3, we can answer the first question. We verified that the implemented solution of the three precision floating-points satisfied the requirements for commercial constrained IoT devices, such as all devices of nRF52 series [20, 21] and all with Direction Finding capability by the time this paper was written. These devices have 192kB to 512kB of flash memory and 24kB to 128kB of RAM. Additionally, we can clearly see in Table 3 that the most time-consuming function relates to the calculation of the covariance matrix (calculate_covmat) which answers the fifth question. Since the IQ matrix ($\mathbf{X}$) is large compared to other matrices, it requires a substantial time to do such computation. Thanks to the optimization described in the previous section, the computation to get eigenvectors, functions power_method, and inverse_power_method, takes a relatively short execution time.

In view of the explanation in the first paragraph of this subsection and the experiment results shown in Table 2, a decrease in floating-point precision does not necessarily decrease execution time and energy consumption, which contradicts what some may expect. Moreover, changing floating-point formats does not affect the accuracy. Thus, answering the third question, although FP16 archives the minimum memory footprint, it is small compared to FP32, but it spends about 64% more execution time and consumes more energy than FP32. So, unless the system requires a very strict memory consumption, FP32 is the best option. The same conclusion applies to the fourth question. That is, FP64 does not only consume about twice of memory, it also spends about 20 times more energy with an execution time 20 times slower than FP32 in exchange for nothing, since it has the same accuracy.

Coin batteries are used for small electronic devices [30], which includes constrained IoT ones. We found that the capacity of such batteries ranges from 1mAh to 2000mAh [7] in a well-established global distributor of semiconductors and electronic components. That means, considering the version FP32 of the method as the only source of energy consumption, the IoT devices can run it from 209643 to 4×10^8 times. Thus, the implemented solution can be used for battery-powered small embedded devices, which answers the second question.

Function	Stack Memory Consumption	Relative Execution Time
calculate_covmat	96B	83.05%
power_method	88B	9.32%
inverse_power_method	48B	-
TLS	88B	7.62%
runESPRIT	80B	-

Table 3: Stack memory and relative execution time values for FP32 ($N = 50$ and $M = 6$).

6 CONCLUSIONS

This paper presented a fast optimized version of Unitary TLS ESPRIT implemented in C99 programming language for low-cost constrained IoT devices where only a single tag transmits a signal at a time due to radio communication system design. To evaluate the viability of the solution, we measured its memory footprint, energy consumption, execution time, and accuracy of three floating-point formats for 6000 DOAs generated by CDL-E channel model with AWGN. The experiments showed the solution fits in commercial embedded devices analyzed by us, notably, Nordic nRF52 and nRF53 series. It was found to be energy-efficient and could attain a sub-millisecond execution time. Additionally, the change of three floating-point precision does not impact its accuracy. Thus, it is possible to employ a small-size floating point precision, however, smaller representations do not necessarily translate into lower execution time, energy consumption, and memory footprint. In fact, precision fully operated by the Floating-Point Unit of the target embedded processors is possibly the best option despite not being the lowest format. In future works, we will use embedded devices equipped with a ULA to get IQ values from real-world scenarios and deal with non-idealities of real antenna arrays.

ACKNOWLEDGMENTS

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] Arm. 2019. *Arm Compiler armclang Reference Guide Version 6.12.* Arm Ltd., Cambridge, England.
[2] Robert Bembenik and Krzysztof Falcman. 2020. BLE Indoor Positioning System Using RSSI-based Trilateration. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 11, 3 (2020), 50–69.
[3] Phakphoom Boonyanant et al. 2004. FPGA Implementation of a Subspace Tracker Based on a Recursive Unitary ESPRIT Algorithm. In *Proc. of IEEE Region 10 Conference TENCON.* IEEE, Chiang Mai, Thailand, 547–550.
[4] Zhizhang Chen, Gopal Gokeda, and Yiqiang Yu. 2010. Chapter 3 - Overview of Basic DOA Estimation Algorithms. In *Introduction to Direction-of-arrival Estimation.* Artech House, Norwood, Massachusetts, United States, 31–63.
[5] Zhizhang Chen, Gopal Gokeda, and Yiqiang Yu. 2010. *Introduction to Direction-of-arrival Estimation.* Artech House, Norwood, Massachusetts, United States.
[6] Pei-Jung Chung, Mats Viberg, and Jia Yu. 2014. DOA Estimation Methods and Algorithms. In *Academic Press Library in Signal Processing.* Vol. 3. Elsevier, Amsterdam, Netherlands, 599–650.
[7] Mouser Electronics. 2022. Coin Cell Battery. https://www.mouser.com/c/power/batteries/coin-cell-battery. [Online; accessed 11-June-2022].
[8] Pranav Kumar Eranti and Buket D Barkana. 2022. An Overview of Direction-of-Arrival Estimation Methods Using Adaptive Directional Time-Frequency Distributions. *Electronics* 11, 9 (2022), 1321.
[9] William Ford. 2015. Chapter 18 - The Algebraic Eigenvalue Problem. In *Numerical Linear Algebra with Applications*, William Ford (Ed.). Academic Press, Boston, 379–438.

[10] William Ford. 2015. Chapter 19 - The Symmetric Eigenvalue Problem. In *Numerical Linear Algebra with Applications*, William Ford (Ed.). Academic Press, Boston, 439–465.

[11] Amos Gilat. 2013. *Numerical Methods for Engineers and Scientists.* Wiley Global Education, New Jersey, United States.

[12] Meng-Chang Hua, Cheng-Han Hsu, and Hsin-Chin Liu. 2012. 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, Poznan, Poland, 1–4.

[13] Charles R Johnson and Roger A Horn. 1985. *Matrix Analysis.* Cambridge University Press, Cambridge, United Kingdom.

[14] Ian Johnson. 2022. *10 Useful Tips for Using the Floating Point Unit on the Cortex-M4.* ARM, Cambridge, United Kingdom. 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 [Online; accessed 12-May-2022].

[15] Nathaniel Johnston et al. 2021. *Advanced Linear and Matrix Algebra.* Springer, Berlin, Germany.

[16] Minseok Kim, Koichi Ichige, and Hiroyuki Arai. 2004. Real-time Smart Antenna System Incorporating FPGA-based Fast DOA Estimator. In *Proc. of 60th Vehicular Technology Conference*, Vol. 1. IEEE, Los Angeles, CA, USA, 160–164.

[17] Ying Liu and Hongyuan Cui. 2015. Antenna Array Signal Direction of Arrival Estimation on Digital Signal Processor (DSP). *Procedia Computer Science* 55 (2015), 782–791.

[18] Ivan Markovsky and Sabine Van Huffel. 2007. Overview of Total Least-Squares Methods. *Signal processing* 87, 10 (2007), 2283–2302.

[19] Nordic Semiconductor. 2021. nRF5340 Product Specification 1.2. https://infocenter.nordicsemi.com/pdf/nRF5340_PS_v1.2.pdf. [Online; accessed 12-May-2022].

[20] Nordic Semiconductor. 2022. Bluetooth Direction Finding. https://www.nordicsemi.com/Products/Bluetooth-Direction-Finding. [Online; accessed 12-May-2022].

[21] Nordic Semiconductors. 2022. nRF52 Series. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52.html. [Online; accessed 12-May-2022].

[22] B. Ottersten, M. Viberg, and T. Kailath. 1991. Performance Analysis of the Total Least Squares ESPRIT Algorithm. *IEEE Transactions on Signal Processing* 39, 5 (1991), 1122–1135. https://doi.org/10.1109/78.80967

[23] Giovanni Pau, Fabio Arena, Yonas Engida Gebremariam, and Ilsun You. 2021. Bluetooth 5.1: An Analysis of Direction Finding Capability for High-Precision Location Services. *Sensors* 21, 11 (2021), 3589.

[24] Justin Solomon. 2015. Chapter 4 - Designing and Analyzing Linear Systems. In *Numerical Algorithms: Methods for Computer Vision, Machine Learning, and Graphics.* CRC press, Florida, United States, 75–76.

[25] N Tayem, M Omer, and A Abul Hussain. 2014. Hardware Implementation of MUSIC and ESPRIT on NI-PXI Platform. In *Proc. of IEEE Military Communications Conference.* IEEE, Baltimore, MD, USA, 329–332.

[26] Nizar Tayem, Syed Ahmed Raza, Muhammad Omer, Mohamed El-Lakki, and Jamal F Nayfeh. 2013. Hardware Implementation of a Proposed Qr-Tls DOA Estimation Method and Music, ESPRIT Algorithms on Ni-Pxi Platform. *Progress In Electromagnetics Research C* 45 (2013), 203–221.

[27] Tiago Troccoli, Juho Pirskanen, Aleksandr Ometov, Jari Nurmi, and Ville Kaseva. 2022. Implementation of Embedded Multiple Signal Classification Algorithm for Mesh IoT Networks. In *2022 International Conference on Localization and GNSS (ICL-GNSS).* IEEE, Tampere, Finland, 1–7. https://doi.org/10.1109/ICL-GNSS54081.2022.9797023

[28] Engin Tuncer and Benjamin Friedlander. 2009. *Classical and Modern Direction-of-Arrival Estimation.* Academic Press, Amsterdam, Netherlands.

[29] Harry L Van Trees. 2004. *Optimum Array Processing: Part IV of Detection, Estimation, and Modulation Theory.* John Wiley & Sons, New Jersey, United States.

[30] Wikipedia contributors. 2022. Button cell — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Button_cell&oldid=1072413928. [Online; accessed 11-June-2022].

[31] Wikipedia contributors. 2022. Eigenvalue algorithm — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Eigenvalue_algorithm. [Online; accessed 11-June-2022].

[32] Martin Woolley. 2021. *Bluetooth Direction Finding: A Technical Overview.* Bluetooth, Cambridge, England.

[33] Arjumand Yaqoob, Umar Farooq, Ghulam Abbas, and Muhammad Usman Asad. 2011. Efficient Hardware Implementation of ESPRIT-likeAlgorithm for Range Estimation of Chirp Spread Spectrum. *International Journal Of Computer and Electrical Engineering* 3, 4 (2011), 473.

[34] Joseph Yiu. 2013. *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors.* Newnes, Cambridge, United Kingdom.