Data in Brief 39 (2021) 107576

Contents lists available at ScienceDirect

Data in Brief

journal homepage: www.elsevier.com/locate/dib

Data Article

Dataset for Bluetooth 5.1 Direction of Arrival with non Uniform Rectangular Arrays

Nicolò Ivan Piazzese[a], Michele Perrone[b], Danilo Pietro Pau[b,*]

[a] *STMicroelectronics, Catania, Italy*
[b] *STMicroelectronics, Agrate Brianza, Italy*

ARTICLE INFO

Article history:
Received 11 June 2021
Revised 4 November 2021
Accepted 9 November 2021
Available online 14 November 2021

Keywords:
Direction of arrival
Bluetooth 5.1
Rectangular antenna arrays
URA8
MUSIC
Pseudo-spectrum

ABSTRACT

This paper presents a dataset for Bluetooth 5.1 direction of arrival (DoA). The dataset was generated with a specifically designed mathematical model of a non-uniform rectangular antenna array. The Python source files that generated the dataset are also provided. The dataset was conceived as a starting point for developing and validating DoA algorithms for real-life scenarios. Unlike other datasets, it contains Bluetooth signals with not only varying intensity of additive white Gaussian noise, but also coherent interfering signals with random DoA coordinates. The dataset is divided into two branches, one consisting of pure sinusoidal tones and the second comprised of baseband Bluetooth signals. Since the codebase which generates the data is included, this dataset has a high reuse potential, and it can be modified to suit also other types of signals or different array topologies.

* Corresponding author.
E-mail addresses: nicolo.piazzese@st.com (N.I. Piazzese), perrone.michele@outlook.com (M. Perrone), danilo.pau@st.com (D.P. Pau).

https://doi.org/10.1016/j.dib.2021.107576

Specifications Table

Subject	Computer Networks and Communications
Specific subject area	Telecommunications and Bluetooth signals
Type of data	Dataset files: *`.csv` files Source code for dataset generation: *`.py` files
How data were acquired	Mathematical simulation of complex signals. Python source files that generated the data are included
Data format	Raw
Parameters for data collection	Useful signal azimuth and elevation (degrees), useful signal to noise ratio (SNR) (dB), useful signal to interfering signals ratio (SIR) (dB), frequency offset between useful signal and interfering signals (interf. offset) (MHz), residual Bluetooth receiver offset (res. offset) (KHz)
Description of data collection	Data was generated with a mathematical model of the signal and of the rectangular antenna array. The simulated signal is a baseband Bluetooth 5.1 signal with constant tone extension. The simulated receiving array is a rectangular array with 8 patch antennas. Many different combinations of data parameter values were considered (SNR, SIR, interf. offset, res. offset)
Data source location	Data was generated at ST Microelectronics, Catania, Italy
Data accessibility	Repository name: Mendeley Data Direct URLs to data: • Part 1: https://doi.org/10.17632/zjn2zxcgyz.1 • Part 2: https://doi.org/10.17632/gysjfpvxpm.1 • Part 3: https://doi.org/10.17632/nnv3vpy4y3.1 • Part 4: https://doi.org/10.17632/362szwsnc4.1 **Please note**: you need to collect all four parts in one folder. Then, you can then use an extraction software of your choice, e.g. 7-Zip (https://www.7-zip.org/), to recreate the entire directory structure of the dataset.

Value of the Data

This dataset simulates the interaction of Bluetooth 5.1 signals with the URA8 array topology. The latest Bluetooth 5.1 standard has introduced important features to aid direction-finding solutions [1]. URA8 is an antenna array composed of eight elements equally spaced on a square edge. This type of array is thoroughly analyzed in [2] and is shown in Fig. 1. The mathematical model of the array is presented in detail in Section 3.1.

- The dataset described in this paper is useful because it provides a valuable tool for the development and validation of DoA algorithms for Bluetooth 5.1 signals. It contains coherent interference with respect to the useful signal, which is always present in real-life scenarios. Since DoA algorithms are usually tested and compared on sinusoidal signals, both sinusoidal signals and Bluetooth signals are included in this dataset. Moreover, the dataset provides the signals in the form of IQ samples, which is how they are outputted by modern Bluetooth 5.1 devices.

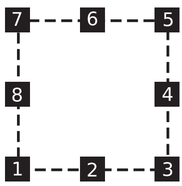

Fig. 1. Geometrical configuration of the non-uniform rectangular array with 8 patch elements.

- Anyone who develops Bluetooth 5.1 tracking applications can benefit greatly from this dataset, because the simulated data are comparable to those obtained through actual measurements. The data have been compared with real-world IQ samples generated by an STMicroelectronics transceiver prototype, fully compliant with the Bluetooth 5.1 standard. This prototype has been equipped with an eight-sensors patch antenna array, with the same topology as the mathematical model of the array shown in Fig. 1 and described in Section 3.1. Experimental results have shown that the measured IQ samples match the signals modeled in the presented dataset with a signal to noise ratio (SNR) of 60dB and a with signal to interference ratio (SIR) below 60dB, showing a mean error equal to 0 and a standard error deviation around 1%. The non-zero value of the standard error deviation is due to a residual stochastic uncompensated frequency offset on the measured IQ samples.
- By including the Python codebase that generates the dataset alongside the dataset itself, the data can be modified and expanded at wish. For example, more than two interfering signals could be added, the topology of the array could be modified, or the number of simulated measurements could be increased. The latter aspect is especially valuable when considering the training of artificial neural networks, which require large quantities of data to generalize appropriately.

1. Data Description

The dataset consists of `.csv` data, along with `.py` Python source files that are used to generate the data. It is divided into two branches, each one corresponding to the type of signal which interacts with the rectangular array. The first branch is devoted to radio frequency (RF) sinusoidal signals, while the second one contains Bluetooth 5.1 signals. Each branch is self-contained within its own folder. The content of the `PureTone_data/` and `BLE_5_1_data/` folders is described in Sections 2.1 and 2.2 respectively. The Python code that was used to generated the dataset, contained in folders `PureTone_code/` and `BLE_5_1_code/`, is described in Section 3.3.

```
/
├── PureTone/
│   ├── PureTone_code/
│   └── PureTone_data/
└── BLE_5_1/
    ├── BLE_5_1_code/
    └── BLE_5_1_data/
```

1.1. RF 2.4GHz pure tone branch

The directory structure is shown below. The data is organized in a series of simulated tests, each contained within its own folder, whose name always begins with `Test_*/`. Each test folder then contains $N_\phi \cdot N_\theta$ sub-folders, where N_ϕ and N_θ are the number of possible azimuth and elevation angles of the useful signal, respectively. The description of every dataset parameter is reported in Table 1, and is valid for both dataset branches. The range of each `PureTone` dataset branch parameter is reported in Table 2.

Table 1
Description of the dataset parameters, valid for both dataset branches.

Parameter	Abbreviation	Description	Measure unit
$<a_i>$	Iof	Frequency offset between the useful signal and interfering signals	MHz
$<b_i>$, $<c_i>$	SIR	Useful signal to interfering signals power ratio	dB
$<d_i>$	SNR	Useful signal to white Gaussian noise power ratio	dB
$<\hat{e}_i>$	rfo	Average residual frequency offset at the Bluetooth receiver	kHz
$<e_i>$	roff	Residual frequency offset at the Bluetooth receiver	hHz
$<f_i>$	CTE	Constant tone extension duration	μs
$<g_i>$, $<h_i>$	cte	Constant tone extension ON/OFF (1/0)	/
$<n_i>$	m or meas	Simulated measurement number	/
$<p_i>$	DR	Data rate	Mbits/s
$<q_i>$	Switching	Switch time between successive antenna elements	μs
$<\texttt{hh}>$, $<\texttt{mm}>$	/	Hours and minutes of data generation	/
$<\texttt{DD}>$, $<\texttt{MM}>$, $<\texttt{YYYY}>$	/	Day, month and year of data generation	/
$<\phi_i^0>$, $<\theta_i^0>$	sig_azim, elv	Azimuth and elevation angle of the useful signal	Degrees ($^\circ$)
$<\phi_i^1>$, $<\theta_i^1>$	iaz, iel	Azimuth and elevation angle of the first interfering signal	Degrees ($^\circ$)
$<\phi_i^2>$, $<\theta_i^2>$	iaz, iel	Azimuth and elevation angle of the second interfering signal	Degrees ($^\circ$)

Table 2
Dataset parameters value range for the PureTone branch.

Parameter	Value range	Measure unit
$<a_i>$	{0, 2}	MHz
$<b_i>$, $<c_i>$	{6, 12}	dB
$<d_i>$	{12, 30, 60}	dB
$<\phi_i>$, $<\theta_i>$	$\phi_i \in [0, 359]$ (1° step), $\theta_i \in [0, 90]$ (3° step)	$^\circ$
$<\phi_i^1>$, $<\theta_i^1>$	Same as $<\phi_i^0>$, $<\theta_i^0>$, random distribution	$^\circ$
$<\phi_i^2>$, $<\theta_i^2>$	Same as $<\phi_i^0>$, $<\theta_i^0>$, random distribution	$^\circ$

PureTone_data/

- Test_Iof<a_1>MHz_SIRs<b_1>dB<c_1>dB_SNR<d_1>dB
- Test_Iof<a_2>MHz_SIRs<b_2>dB<c_2>dB_SNR<d_2>dB
- Test_Iof<a_3>MHz_SIRs<b_3>dB<c_3>dB_SNR<d_3>dB
- ...
- Test_Iof<a_N>MHz_SIRs<b_N>dB<c_N>dB_SNR<d_N>dB
  - sig_azim0_elv0
  - sig_azim0_elv3
  - sig_azim0_elv6
  - ...
  - sig_azim0_elv87
  - sig_azim1_elv0
  - sig_azim1_elv3
  - sig_azim1_elv6
  - ...
  - sig_azim1_elv87
  - sig_azim2_elv0
  - sig_azim2_elv3
  - sig_azim2_elv6
  - ...
  - sig_azim<ϕ_i>_elv<θ_i>
  - ...
  - sig_azim359_elv87
    - doares_saz359_sel87.csv
    - info_saz359_selv87_<hh>_<mm>_of_<DD>_<MM>_<YYYY>.csv
    - X_m<n_0>_saz359_sel87_iaz[<ϕ_1^1> <ϕ_1^1>]_iel[<θ_1^1> <θ_1^2>].csv
    - X_m<n_0>_saz359_sel87_iaz[<ϕ_2^1> <ϕ_2^1>]_iel[<θ_2^1> <θ_2^2>].csv
    - X_m<n_0>_saz359_sel87_iaz[<ϕ_3^1> <ϕ_3^1>]_iel[<θ_3^1> <θ_3^2>].csv
    - X_m<n_1>_saz359_sel87_iaz[<ϕ_4^1> <ϕ_4^1>]_iel[<θ_4^1> <θ_4^2>].csv
    - X_m<n_1>_saz359_sel87_iaz[<ϕ_5^1> <ϕ_5^1>]_iel[<θ_5^1> <θ_5^2>].csv
    - X_m<n_1>_saz359_sel87_iaz[<ϕ_6^1> <ϕ_6^1>]_iel[<θ_6^1> <θ_6^2>].csv

There are three types of `.csv` files for each `sig_azim`$<\phi_i>$`_elv`$<\theta_i>$ folder, all of them using `UTF-8` encoding and comma as separator. **The following description is valid for both dataset branches:**

- `doares_saz`$<\phi_i>$`_sel`$<\theta_i>$`.csv` files contain the DoA of the useful signal, estimated by the Multiple signal classification algorithm (MUSIC), which is described in Section 3.2. They are meant to serve as a ground truth for performance comparison with other DoA algorithms. An example of such file is shown in Table 3. The column headers and their abbreviations correspond to the parameters described in Table 1. The results obtained by MUSIC are contained in `res_azim` (resulting azimuth) and `res_elev` (resulting elevation) columns.
- `info_saz`$<\phi_i>$`_sel`$<\theta_i>$`_`<hh>`_`<mm>`_of_`<DD>`_`<MM>`_`<YYYY>`.csv` files are useful in the case that one does not want to rely on the hierarchy of the directory tree to read the key dataset parameters. Their generic structure is shown in Table 4. They contain the useful signal angular position range (minimum, maximum, and step), the position of the useful signal, the frequency offset $< a_i >$, the signal to noise ratio $< d_i >$ and the first interfering signal to useful signal ratio $< b_i >$. The parameters correspond to those described in Table 1.

Table 3
Example of a doares_saz$<\phi_i>$_sel$<\theta_i>$.csv file.

meas	sig_azim	sig_elev	CTE	iaz	iel	res_azim	res_elev
0	0.000000	0.000000	[0 0]	[305 16]	[16 44]	324.000000	9.000000
1	0.000000	0.000000	[0 0]	[305 16]	[16 44]	321.000000	5.000000
0	0.000000	0.000000	[1 1]	[292 169]	[44 44]	292.000000	6.000000
1	0.000000	0.000000	[1 1]	[292 169]	[44 44]	273.000000	9.000000
0	0.000000	0.000000	[1 1]	[337 355]	[21 33]	329.000000	9.000000
1	0.000000	0.000000	[1 1]	[337 355]	[21 33]	345.000000	9.000000

Table 4
Generic content of info_saz$<\phi_i>$_sel$<\theta_i>$_<hh>_<mm>_of_<DD>_<MM>_<YYYY>.csv files.

Test started at <hh> <mm> <DD> <MM> <YYYY>	
Useful signal	
useful signal test ranges	azimuth [0 359 1] elevation [0 90 3]
useful signal position	azimuth $<\phi_i^0>$ elevation $<\theta_i^0>$
Interfering signals	
first interf. signal frequency offset from useful signal	$<a_i>$ *MHz*
SNR and SIR	
SNR	$<d_i>$ *dB*
first interf. signal SIR	$<b_i>$ *dB*

- X_m< n_i >_*.csv files contain the 8×70 X_{IQ} complex-valued IQ matrices, which result from the interaction of the signals and the rectangular antenna array. These files do not contain any header. Each 8×1 column is a complex-valued IQ vector $\boldsymbol{x}_{IQ}{}^T$. The complex numbers are in the format shown in Table 5

Table 5
Example of a complex-valued IQ vector $\boldsymbol{x}_{IQ}{}^T$ contained in X_m<n_i>_*.csv files.

1.859307540261690983e+00-1.614217293859837865e-01j
1.483198132423238835e+00-3.605448754221074470e-01j
9.662254468967348409e-01-4.818835170878809637e-01j
1.644941300650671900e+00-1.455036290828832235e-01j
8.238801969038891393e-01+5.024682120440079336e-01j
9.591105763646003979e-01-6.146509479381243035e-02j
9.636106402553353822e-01-2.972847799272813618e-01j
9.085768220898262637e-01+5.888629266394146411e-01j

1.2. Baseband Bluetooth 5.1 signal branch

The directory structure is very similar to the PureTone branch and is reported below. The difference consists in the greater number of parameters due to the presence of the residual frequency offset and the constant tone extension (CTE), which are typical of Bluetooth 5.1 signals. Table 1 provides a description for each parameter, while Table 6 reports the value range that

Table 6
Dataset parameters value range for the BLE_5_1 branch.

Parameter	Value range	Measure unit
$<a_i>$	{0}	MHz
$<b_i>$, $<c_i>$	{3, 6, 12, 18, 24, 30, 36, 48, 54, 60}	dB
$<d_i>$	{30, 48, 60}	dB
$<\hat{e}_i>$	{-3, 0, +3}	kHz
$<e_i>$	$e_i \in [-3, +3]$, Gaussian distribution	hHz
$<f_i>$	{1200}	μs
$<g_i>$, $<h_i>$	{0, 1} (OFF/ON)	/
$<p_i>$	{1}	Mbit/s
$<q_i>$	{1, 2}	μs
$<\phi_i>$, $<\theta_i>$	$\phi_i \in [0, 359]$ (1° step), $\theta_i \in [0, 90]$ (3° step)	°
$<\phi_i^1>$, $<\theta_i^1>$	Same value range as $<\phi_i^0>$, $<\theta_i^0>$, random distribution	°
$<\phi_i^2>$, $<\theta_i^2>$	Same value range as $<\phi_i^0>$, $<\theta_i^0>$, random distribution	°

can be assumed by each parameter. The three types of `.csv` files within each `sig_azim<`ϕ_i`>_elv<`θ_i`>` folder are identical with respect to the `PureTone` dataset branch.

```
BLE_5_1_data/
└── DR<p_i>Mbps_Switching<q_i>us
    ├── Test_Iof<a_1>MHz_SIRs<b_1>dB<c_1>dB_SNR<d_1>dB_rfo<e_1>KHz_CTE<f_1>us
    ├── Test_Iof<a_2>MHz_SIRs<b_2>dB<c_2>dB_SNR<d_2>dB_rfo<e_2>KHz_CTE<f_2>us
    ├── Test_Iof<a_3>MHz_SIRs<b_3>dB<c_3>dB_SNR<d_3>dB_rfo<e_3>KHz_CTE<f_3>us
    ├── ...
    └── Test_Iof<a_N>MHz_SIRs<b_N>dB<c_N>dB_SNR<d_N>dB_rfo<e_N>KHz_CTE<f_N>us
        ├── sig_azim0_elv0
        ├── sig_azim0_elv3
        ├── sig_azim0_elv6
        ├── ...
        ├── sig_azim0_elv87
        ├── sig_azim1_elv0
        ├── sig_azim1_elv3
        ├── sig_azim1_elv6
        ├── ...
        ├── sig_azim1_elv87
        ├── sig_azim2_elv0
        ├── sig_azim2_elv3
        ├── sig_azim2_elv6
        ├── ...
        ├── sig_azim<φ_i>_elv<θ_i>
        ├── ...
        └── sig_azim359_elv87
            ├── doares_saz359_sel87.csv
            ├── info_saz359_selv87_<hh>_<mm>_of_<DD>_<MM>_<YYYY>.csv
            ├── X_m0_saz359_sel87_cte[<g_1> <h_1>]_iaz[<φ_1^1> <φ_1^1>]_iel[<θ_1^1> <θ_1^2>]_roff<m_1>hHz.csv
            ├── X_m0_saz359_sel87_cte[<g_2> <h_2>]_iaz[<φ_2^1> <φ_2^1>]_iel[<θ_2^1> <θ_2^2>]_roff<m_2>hHz.csv
            ├── X_m0_saz359_sel87_cte[<g_3> <h_3>]_iaz[<φ_3^1> <φ_3^1>]_iel[<θ_3^1> <θ_3^2>]_roff<m_3>hHz.csv
            ├── X_m1_saz359_sel87_cte[<g_4> <h_4>]_iaz[<φ_4^1> <φ_4^1>]_iel[<θ_4^1> <θ_4^2>]_roff<m_4>hHz.csv
            ├── X_m1_saz359_sel87_cte[<g_5> <h_5>]_iaz[<φ_5^1> <φ_5^1>]_iel[<θ_5^1> <θ_5^2>]_roff<m_5>hHz.csv
            └── X_m1_saz359_sel87_cte[<g_6> <h_6>]_iaz[<φ_6^1> <φ_6^1>]_iel[<θ_6^1> <θ_6^2>]_roff<m_6>hHz.csv
```

2. Experimental Design, Materials and Methods

The dataset is based on a mathematical model of a non-uniform rectangular antenna array with 8 elements. The following sections provide a description of the mathematical model of the array, of the impinging signals, and of their Python implementation which is provided with the dataset.

2.1. Mathematical model

https://www.overleaf.com/project/617a8bcbbeb847848f1fd78c The array model corresponds a rectangular array of $N = 8$ numbered antenna elements, evenly spaced at a distance $d = \frac{\lambda}{2.5}$ on a square edge, as shown in Fig. 1. Because of the lack of a central element, this array is non-uniform.

The rectangular array receives M signals $s_i(n)$, incident with angles

$$(\theta_1, \phi_1), (\theta_2, \phi_2), ..., (\theta_m, \phi_m), ..., (\theta_M, \phi_M) \tag{1}$$

where θ_m and ϕ_m are the m-th elevation and azimuth angles of each signal. The signals have also a defined power $P_1, P_2, P_m, ..., P_M$.

The array response matrix A describes the interaction of the impinging signals with the rectangular antenna array. For the topology of the array shown in Fig. 1, the array response matrix is

$$A = [\boldsymbol{a}(\theta_1, \phi_1)^T, \boldsymbol{a}(\theta_2, \phi_2)^T, ..., \boldsymbol{a}(\theta_M, \phi_M)^T)] \tag{2}$$

where the array response vectors $\boldsymbol{a}(\theta, \phi)$ are defined as:

$$\boldsymbol{a}(\theta, \phi) = \begin{bmatrix} 1 \\ e^{-j\gamma(\theta)(dcos(\phi))} \\ e^{-j\gamma(\theta)(d2cos(\phi))} \\ e^{-j\gamma(\theta)(2dcos(\phi)+dsin(\phi))} \\ e^{-j\gamma(\theta)(2dcos(\phi)+2dsin(\phi))} \\ e^{-j\gamma(\theta)(dcos(\phi)+2dsin(\phi))} \\ e^{-j\gamma(\theta)(d2sin(\phi))} \\ e^{-j\gamma(\theta)(dsin(\phi))} \end{bmatrix} \tag{3}$$

with $d = \frac{\lambda}{2.5}$ and

$$\gamma(\theta) = 2\pi sin(\theta) \tag{4}$$

As mentioned in Section 2, the dataset is subdivided into two branches. For the **Bluetooth 5.1 dataset branch**, the signals $s_i(n)$ are a baseband model of the signals at the analog to digital converter (ADC) of the Bluetooth receiver. Apart from the frequency offset, the carrier translation and the RF impairments are negligible for the study of the DoA. Regarding the frequency offset between the transmitter and the receiver, the majority of the offset is corrected at the automatic frequency corrector (AFC). However, a small part of this offset typically remains, which we indicate as f_{off}. This residual offset is able to impact negatively the estimation of the DoA. Our mathematical model of BLE 5.1 signals is that of binary Gaussian frequency shift keying (GFSK) modulated signals:

$$s_i(n) = e^{j\cdot\alpha(n)} \cdot e^{(2\cdot\pi\cdot f_{off}\cdot n)/F_s} \tag{5}$$

$$\alpha(n) = \frac{2\pi \cdot F_{dev}}{F_s} \cdot \sum_{k=0}^{n} \sum_{i} b_i p(k - iN_s) \tag{6}$$

$$F_{dev} = \frac{DR \cdot h}{2} \tag{7}$$

where DR is the data rate, f_{off} is a possible frequency offset error, $F_s = 16MHz$ is the sampling frequency at the receiver's ADC, n represents the discrete-time index, h is the modulation index, $p(n)$ is the symbol pulse as defined in [3], and $b_i \in \{1, -1\}$ is the binary symbol to be transmitted. As a mathematical model for the baseband CTE, Eq. 5 with all binary symbols b_i equal to 1 is used.

The baseband samples at the Bluetooth receiver, running at a proper sample period T_s without ADC impairments can be expressed as:

$$X_{T_s} = A \cdot \begin{bmatrix} P_1 & 0 & ... & 0 \\ 0 & P_2 & ... & 0 \\ 0 & ... & ... & 0 \\ 0 & 0 & ... & P_M \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ ... \\ s_M \end{bmatrix} + \boldsymbol{n} \tag{8}$$

where $s_1, s_2, ..., s_M$ are the complex impinging signals, generated by using Eq. 5 with a sampling period T_s and $\boldsymbol{n} = [n_1, n_2, ..., n_8]^T$ is the noise vector at the analog to digital converter (ADC), with zero mean and variance σ^2, which is related to the SNR of the useful tag signal. The noise vector is comprised of AWGN noise, which approximates the noise of the receiver, and does not take into account the interfering signals. When one of the s_i signals is a CTE, all binary symbols are set to 1, otherwise they are selected randomly.

Considering the IQ sampling process of the Bluetooth receiver and calling T_{switch} and $T_{sample} = 1/F_s$ the switch slot duration and the sampling slot duration respectively, both set to 1 μs or 2 μs, the X_{T_s} data are down-sampled with a factor of $round((T_{switch} + T_{sample})/T_s)$, thus obtaining the IQ samples in form of X_{IQ}, which is a complex matrix with size $N \times N_{samp}$.

Because of the filtering chain, the power of the adjacent channels, alternate channels and all remaining channels is strongly attenuated in comparison to the reflections of useful signal. This means that, in the BLE 5.1 model, all of the interfering signals are in fact the reflections of the useful signal itself.

In the case of **pure tone dataset branch**, the signals $s_i(n)$ are defined as

$$s_i(n) = e^{j \cdot 2\pi \cdot (F_c + ch) \cdot (n/F_s)} \tag{9}$$

where $F_c = 2.4GHz$, ch is the channel spacing (which is an integer multiple of $2MHz$), and F_s is the sampling frequency, set to $F_s = F_c \cdot 32$. In this model, we do not decimate the s_i signals.

The IQ samples are obtained as

$$X_{IQ} = A \cdot \begin{bmatrix} P_1 & 0 & \dots & 0 \\ 0 & P_2 & \dots & 0 \\ 0 & \dots & \dots & 0 \\ 0 & 0 & \dots & P_M \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ \dots \\ s_M \end{bmatrix} + \boldsymbol{n} \tag{10}$$

where $\boldsymbol{n}$ is the AWGN noise at the antenna elements. Please note that the noise $\boldsymbol{n}$ which is added in the pure signal model is much more wideband when compared to the $\boldsymbol{n}$ noise of the Bluetooth model, because the BLE model is a baseband model, while the pure tone signal is an RF model. Therefore, the signal to noise ratios (SNRs) obtained with the two models are not comparable. This is because the SNR depends on the bandwidth of the noise channel.

2.2. The MUSIC algorithm

In addition to the modeled RF 2.4GHz and Bluetooth 5.1 signals, this dataset provides a ground truth for the development and testing of different algorithms. For this purpose, the Multiple signal classification (MUSIC) algorithm [4] was implemented and applied to the simulated dataset signals. MUSIC is a signal subspace method for DoA estimation, which exploits the eigen-structure of the autocorrelation matrix of the signal to find the signal sub-space and noise sub-space. The MUSIC pseudo-spectrum $P(\phi, \theta)$ is obtained in following way:

$$P(\phi, \theta) = \frac{1}{\boldsymbol{a}(\phi, \theta)^H \cdot Q_n \cdot Q_n^H \cdot \boldsymbol{a}(\phi, \theta)} \tag{11}$$

where $\boldsymbol{a}(\theta, \phi)$ are the array response vectors defined in Eq. 3 and Q_n the eigen vectors of the noise sub-space. The DoA estimate of the source signal is then obtained by finding the peak in the above defined pseudo-spectrum.

2.3. Python implementation

Note on licensing. The Python implementation that is shipped with this dataset is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. If you make improvements to the code, you are invited to share those changes with the community.

File structure and dependencies. The Python implementation is written in Python 3 and is contained inside the following *.py files:

```
/
├── README.txt
├── PureTone/
│   ├── PureTone_code/
│   │   ├── algorithmics_URA8.py
│   │   ├── e2e_URA8_PureTone.py
│   │   ├── e2e_URA8_PureTone_DOAview.py
│   │   └── e2e_URA8_PureTone_DSgen.py
│   └── PureTone_data/
└── BLE_5_1/
    ├── BLE_5_1_code/
    │   ├── algorithmics_URA8.py
    │   ├── e2e_URA8_BLE.py
    │   ├── e2e_URA8_BLE_DOAview.py
    │   └── e2e_URA8_BLE_DSgen.py
    └── BLE_5_1_data/
```

The following dependencies are needed: `os`, `time`, `numpy`, `scipy` and `matplotlib`.

The following paragraphs describe the content of each Python module by making also reference to the mathematical equations of the previous sections.

The algorithmic module. The `algorithmics_URA8.py` module is common to both branches of the dataset. It contains the definition of the conventional steering vector for URA8 as defined in Eq. 3 and the implementation of the MUSIC algorithm as described in Section 3.2. The function which generates the steering vector is the following:

```
# Conventional steering vector for URA8  ref. eq. (3)
def CSV_URA8(Nant,dx,dy,apx,apy,phi,theta):
    gamma = np.sin(theta)
    SS = np.zeros((Nant), dtype=complex)
    for nx in range(Nant):
        SS[nx] = np.exp(-1j*gamma*2*np.pi*(dx*(apx[nx])*np.cos(phi)+dy*(apy[nx])*np.sin(phi)));

    return SS
```

The URA8 steering vector from Eq. 3 is also defined inside the MUSIC implementation:

```
                #STeering vector for URA8 ref. eq(3)
              SV = np.array([1.0+1j*0.0,
                             np.exp(-1j*gamma_vec[idy]*(dxcos_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(2*dxcos_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(2*dxcos_phi[idx]+dysin_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(2*dxcos_phi[idx]+2*dysin_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(dxcos_phi[idx]+2*dysin_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(2*dysin_phi[idx])),
                             np.exp(-1j*gamma_vec[idy]*(dysin_phi[idx]))]);
```

The MUSIC pseudo-spectrum from Eq. 11 is then estimated:

```
            if np.isscalar(Qn2) == True:

                SVH = SV.conj().T
                numerator = np.matmul(SVH,SV)
                Qn2SV = Qn2*SV
                denominator = np.matmul(SVH,Qn2SV)

            else:
                SVH = SV.conj().T
                numerator = np.matmul(SVH,SV)
                denominator = np.matmul(SVH,np.matmul(Qn2,SV))

            PP = numerator/denominator # ref. eq.(11)
```

The signal model modules. For the Bluetooth 5.1 branch, the signals are modeled inside the `e2e_URA8_BLE.py` module. The frequency deviation from Eq. 7 is implemented as

```
    fdev = data_rate*mod_index/2.0; #frequency deviation  ref. eq.(7)
```

The baseband signals $s_i(n)$ with the frequency offset f_{off}, as defined in Eqs. 5 and 6, are implemented as

```
    freqoff = Ch_num    # CHannel offset (in Hz)
                          #numer of channels BLE
    BB_ModulatedSignal = np.exp(1j*(2.0*np.pi*np.cumsum(fdev*pckt_upsampled))/(Fbb));   #
     signal in baseband   ref. eq. (5) and eq. (6)
    len_BB = np.size(BB_ModulatedSignal, 0)
    timex = np.arange(0,len_BB)/Fbb                                                        #
     time axis
    y_tmp = np.exp(1j*2.0*np.pi*timex*freqoff)                                            #
     add frequency offset.  ref. eq.(5)
    BB_ModulatedSignal = y_tmp*BB_ModulatedSignal                                         #
     signal in baseband for every channel
```

The directions of the incident signals from Eq. 1 and the array response vectors from Eq. 3 are generated in

```
    # build the A matrix                               ref. eq.(1)
    for h in range(len(a_phi)):
        theta = a_theta[h]*np.pi/180.0
        phi = a_phi[h]*np.pi/180.0;
        a_tmp = algo.CSV_URA8(N,d,d,apx,apy,phi,theta)  #ref.eq. (3)
```

The array response matrix from Eq. 8 is implemented in the following lines:

```
    for meas in range(measures):
        noise = np.sqrt(noise_energy/2)*(np.random.randn(N, samples) +1j*np.random.randn(N,
      samples)); #Uncorrelated noise
        AdiagVrx = np.matmul(A,diagVrx)
        AdiagVrxsig = np.matmul(AdiagVrx,rx_sig)
        X = (AdiagVrxsig+noise); # X matrix      ref. eq. (8)
        X = X[:,::down_iq]    #donw-sampling the ADC signal.
                                                 #Decimation after adc
        X = X[:, gauss_del:]  #remove the gauss filter delay
                                             #Remove signals delayed due to Gaussian
     filter
```

For the PureTone branch, the signals are modeled inside the `e2e_URA8_PureTone.py` module. The signals $s_i(n)$ from Eq. 9 are implemented as follows:

```
# build the signal matrix
for idx in range(len(a_phi)):
    #  stack in a tx signal matrix all trasmitted signals
    if (idx == 0):
        tx_sig = np.exp(1j*2*np.pi*Fsin[idx]*t);          #test signal ref. eq.(9)
        rx_sig = tx_sig

    else:
        tx_sig = np.exp(1j*2*np.pi*Fsin[idx]*t);          #test signal ref. eq.(9)
        rx_sig = np.vstack((rx_sig,tx_sig))
```

The array response vector from Eq. 3, which comprise the array response matrix from Eq. 2 are then computed in the following lines:

```
 # build the A matrix
for h in range(len(a_phi)):
    theta = a_theta[h]*np.pi/180.0
    phi = a_phi[h]*np.pi/180.0;
    a_tmp = algo.CSV_URA8(N,d,d,apx,apy,phi,theta)  #ref. eq.(3)

    if(h == 0):
        A = a_tmp
    else:
        A = np.vstack((A,a_tmp))
```

The X_{IQ} sample matrix from Eq. 10 is then obtained:

```
A = np.transpose(A)
Pw = 10.0**(Pw_dB/20.0)
diagVrx = np.diag(Pw)
for meas in range(measures):
    noise = np.sqrt(noise_energy/2)*(np.random.randn(N, samples) +1j*np.random.randn(N,
  samples)); #Uncorrelated noise
    AdiagVrx = np.matmul(A,diagVrx)
    AdiagVrxsig = np.matmul(AdiagVrx,rx_sig)
    X = (AdiagVrxsig+noise); # X matrix             ref. eq(10)
```

The dataset generator modules. The dataset is generated by the `e2e_URA8_BLE_DSgen.py` and the `e2e_URA8_PureTone_DSgen.py` modules. Both modules do not accept console arguments. The Bluetooth 5.1 dataset generator allows for the setting of the following parameters before being launched:

```
interf_Ntests = 3  # number of test with different interfers positions for each tag azimuth and elevation couple.
measures = 2 #number of measures of IQ samples tfor each Interf. tests
save_opt = 3 #saving options

N = 8 # is the antenna number
# BLE 5.1 settings
data_rate = 1e6 # data rate (or the symbol rate) in bps
t_slot_duration = 2*1e-6 # slot duration of the IQ sampling ( in sec)
CTETime = 150*N # CTE time (ref. to all antennas) in microsec

# URA settings
SNR = 60 # SNR of the AWGN added (refered to ADC) in dB
max_freq_offset_res= 0 #this is the resiual frequency offset between the tx tag and rx (considering that a large amount is corrected by  AFC inside the BLE demodulator)

saz_start = 0 # azimuth start angle of the tag signal
saz_end = 360 #azimuth end angle of tag signal
saz_step= 1 #azimuth step angle of tag signal

sel_start = 0 #elevation start angle of tag signal
sel_end = 90 #elevation end angle of tag signal
sel_step = 3  #elevation step angle of tag signal

SIR1_tests = np.array([3, 6, 12, 18, 30, 48, 60, 3, 6, 12, 18, 30, 48, 54,]) #SIR is the Signal to Interference ration (in dBc) SIR1 is refered to the first interf.
SIR2_tests = np.array([3, 6, 12, 18, 30, 48, 60, 6, 12, 18, 24, 36, 54, 60,]) #SIR2 is refered to the 2th interf
```

A similar approach is used for the parameters of the PureTone dataset generator:

```
Nsamp = 70   #samples of the tone
interf_Ntests = 3  # number of test with different interfers positions for each tag azimuth and elevation couple.
measures = 2 #number of measures of IQ samples tfor each Interf. tests
save_opt = 3 #saving options
SNR = 60    #SNR of the AWGN (dB)

saz_start = 0 # azimuth start angle of the tag signal
saz_end = 360 #azimuth end angle of tag signal
saz_step= 1 #azimuth step angle of tag signal

sel_start = 0 #elevation start angle of tag signal
sel_end = 90 #elevation end angle of tag signal
sel_step = 3  #elevation step angle of tag signal

SIR1_tests = np.array([3, 6, 12, 18, 30, 48, 60, 3, 6, 12, 18, 30, 48, 54,]) #SIR is the Signal to Interference ration (in dBc) SIR1 is refered to the first interf.
SIR2_tests = np.array([3, 6, 12, 18, 30, 48, 60, 6, 12, 18, 24, 36, 54, 60,]) #SIR2 is refered to the 2th interf
```

The testing modules. The `e2e_URA8_BLE_DOAview.py` and `e2e_URA8_PureTone_DOAview.py` modules are meant as an aid for the setting of the dataset parameters prior to its actual generation. Their output is a 3D plot of the MUSIC pseudo-spectrum. The following code is taken from the e2e_URA8_BLE_DOAview.py module. It gives the possibility of manually setting the following parameters:

```
Fadc = 16e6  #reference frequency of the model
Fs = Fadc    #frequency sampling
N = 8        # number of antennas
M = 1        # number of signal to detect
d=1/2.5      #antennas distances (in lambda)

# BLE 5.1 settings
data_rate = 1e6                                  # data rate (or the symbol rate) in bps
t_slot_duration = 2*1e-6                         # slot duration of the IQ sampling ( in sec
     )
t_iq_samp= 2*t_slot_duration                     #iq sampling alternating slots
f_iq_samp=1/t_iq_samp                            #iq samples in freq
down_iq=int(np.round(Fs/f_iq_samp))              #down sampling iq

CTETime = 150*N                       # CTE time (ref. to each antenna) in microsec
max_freq_offset_res=0e3               #residual frequency offset (rfo)
freq_offset_res=np.random.rand(1,1)
freq_offset= max_freq_offset_res*(2*freq_offset_res[0]-1)
fc=0 +  freq_offset                                    # Carrier frequency of CTE
                                                       #Carrier frequency of CTE



# URA settings
SNR = 60                                # SNR of the AWGN added (refered to ADC) in dB
a_phi = np.array([70, 18, 155]);        # azimuthal angles of arrival under test (dimension
      1xM)
a_theta = np.array([45, 75, 75])        #elevation angles of arravial
Pw_dB = np.array([0, -6, -6]);          # Rx useful power (dBm)  of the  received signal (
     dimension 1XM)
Ch_freq=np.array([fc, fc, fc]);         #signals frequency
a_CTE_on= np.array([1,1,1])             #1 =BLE 5.1 CTE, 0 generic GFSK signal
ovs = int(Fadc/data_rate);              # upsampling factor

mod_index = 0.5;                        #modulation index of BLE GFSK signal.between
     0.45-0.55
mod_type = 1;                           #GFSK=1 FSK=0 modulation type
gauss_del = 2                             #expected gaussian delay
apx = np.array([0, 1, 2, 2, 2, 1, 0, 0]) #array of position index on x direction
apy = np.array([0, 0, 0, 1, 2, 2, 2, 1]) #array of position index on y direction
SNR_dB = SNR                #Signal Noise to ratio (in dB)
noise_energy = 10**((np.max(Pw_dB)-SNR_dB)/10.0) #noise energy
```

The e2e_URA8_PureTone_DOAview.py gives the possibility of manually setting the following parameters:

```
M = 1                                              #number of useful impinging signals
N = 8                                                  #number of antennas
fc = 2.4e9;                                            # Carrier frequency of the Pure tone
d = 1/2.5                                              #antennas distances (in lambda)
Fs = fc*32;                                              # frequency sampling
samples = 100;                                        #samples at each antenna

a_phi = np.array([30, 78, 155]);        # azimuthal angles of arrival under test (dimension
      1xM)
a_theta = np.array([45, 75, 75])        #elevation angles of arravial
Pw_dB = np.array([0, -12, -30]);          # Rx useful power (dBm)  of the  received signal
     (dimension 1XM)

Fch = np.array([0, 2e6, 500e3])                                   #signals frequency
Fsin = fc+Fch;                                                  #signals frequency
apx = np.array([0, 1, 2, 2, 2, 1, 0, 0]) #array of position index on x direction
apy = np.array([0, 0, 0, 1, 2, 2, 2, 1]) #array of position index on y direction
SNR_dB = 60                 #Signal Noise to ratio (in dB)
noise_energy = 10**((np.max(Pw_dB)-SNR_dB)/10.0) #noise energy
t = np.arange(0,samples)*1.0/Fs;                 # Time
```

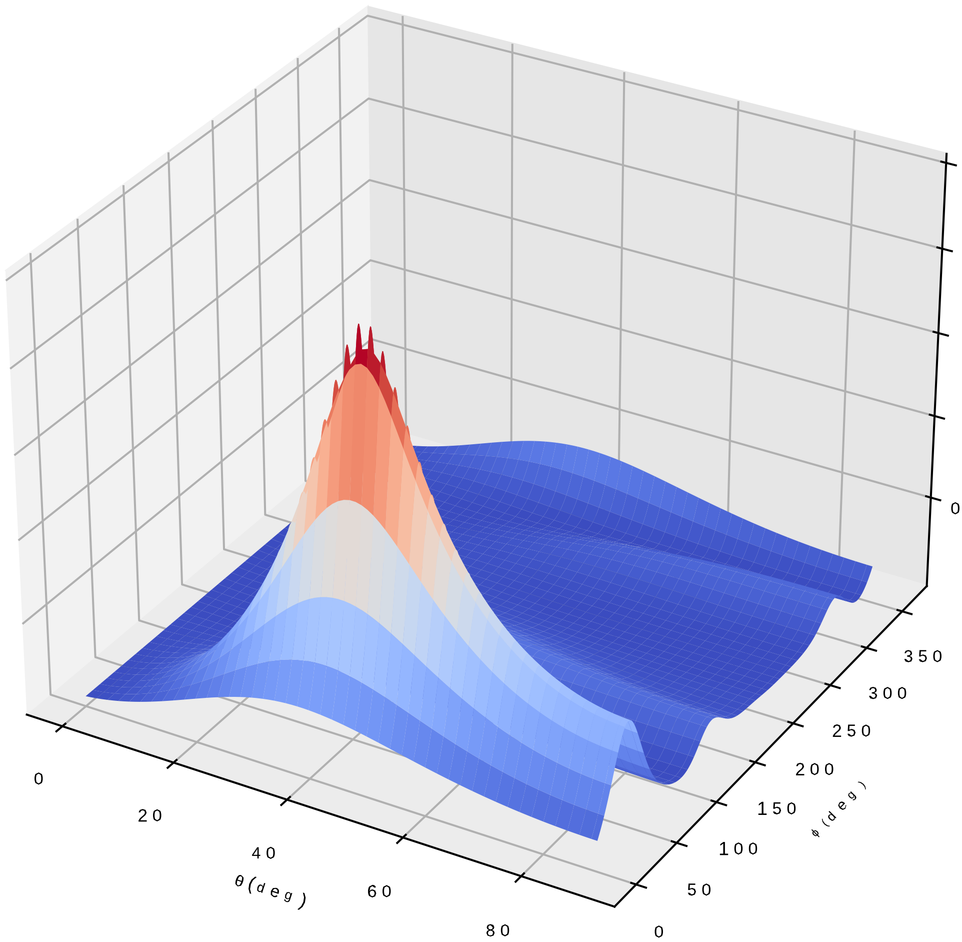

Fig. 2. MUSIC pseudo-spectrum plotted by the DoA view modules.

An example of the plots created by the modules is shown in Fig. 2.

Ethics Statement

The authors declare that the data presented in this article did not involve any use of human subjects, animal experiments nor data collected from social media platforms.

CRediT Author Statement

Nicolò Ivan Piazzese: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft, Writing – review & editing; **Michele Perrone:** Conceptualization, Validation, Data curation, Writing – original draft, Writing – review & editing; **Danilo Pietro Pau:** Conceptualization, Validation, Writing – original draft, Writing – review & editing, Supervision, Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships which have, or could be perceived to have, influenced the work reported in this article.

References

[1] G. Pau, F. Arena, Y.E. Gebremariam, I. You, Bluetooth 5.1: An analysis of direction finding capability for high-precision location services, Sensors 21 (11) (2021), doi:10.3390/s21113589. https://www.mdpi.com/1424-8220/21/11/3589.

[2] L. Jin, L. li, H. Wang, Investigation of different types of array structures for smart antennas, in: 2008 International Conference on Microwave and Millimeter Wave Technology, volume 3, 2008, pp. 1160–1163, doi:10.1109/ICMMT.2008.4540633.

[3] T. Rappaport, Wireless communications: Principles and practice, Prentice Hall communications engineering and emerging technologies series, 2nd, Prentice Hall, 2002. Includes bibliographical references and index

[4] P. Stoica, A. Nehorai, Music, maximum likelihood and cramer-rao bound, in: ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing, volume 4, 1988, pp. 2296–2299, doi:10.1109/ICASSP.1988.197097.