RFChallenge:

Collision of the Drones: A Challenge in Multi-Sensor Signal Separation

Guide for Challenge Participants

Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

I. Overview

In the single-channel portion of *RFChallenge*, the task is to design an algorithm which learns and exploits the waveform structure in co-channel RF signals to separate them. When available, multi-channel measurements offer additional leverage for signal separation in the form of low-dimensional spatial structure. Namely manmade interference typically arises from point RF emitters, and point emitters are intrinsically low-dimensional when observed by an antenna array with a sufficient number of elements[1]. In this part of the Challenge, the ISM[2]-band downlink of an unmanned aerial vehicle (UAV) is buried in co-channel interference from another UAV at a distinct direction-of-arrival. The task in this part of the Challenge is to exploit the structure in multi-channel measurements of the mixture of the two UAV waveforms to extract the downlink signal from its interference. The lack of prior information on either the individual waveforms, or their respective array responses at the receiver, makes unsupervised machine learning an intriguing framework for solving this blind source separation problem.

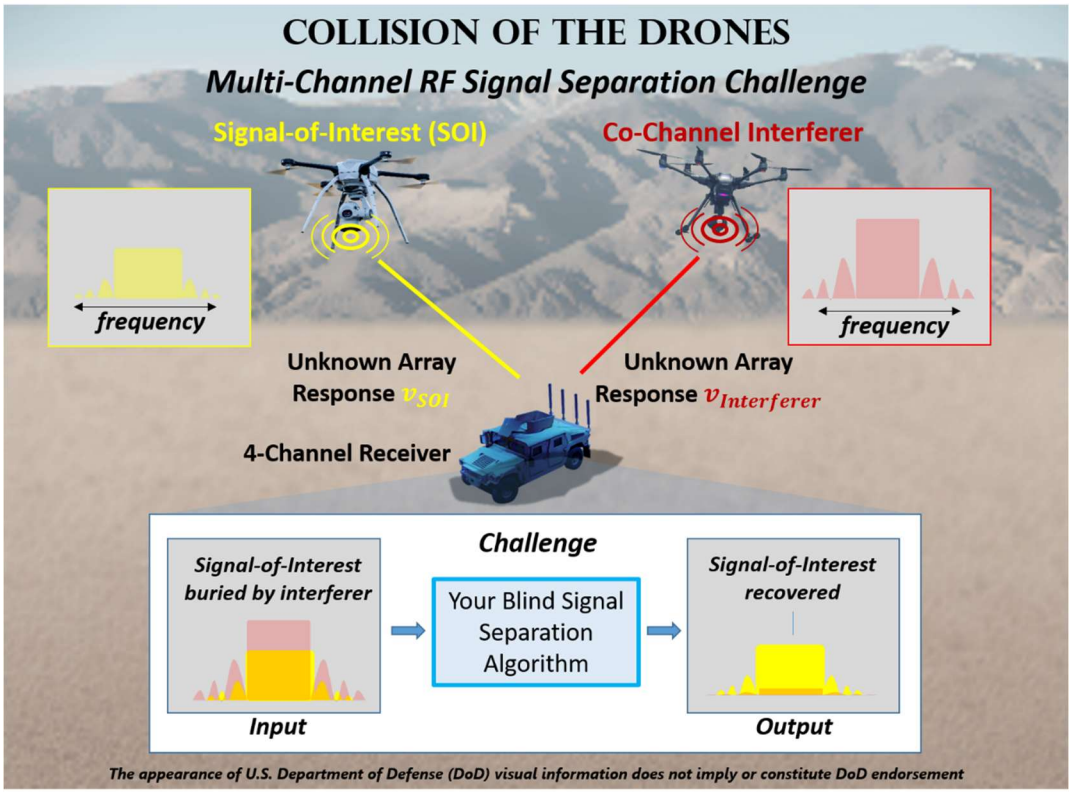

Figure 1: Problem Overview

[1] The *M*-dimensional measurements captured by an *M*-element antenna array can be represented as linear combinations of $N < M$ array responses associated with the directions at which each of the *N* point emitter signals arrives at the array

[2] Industrial, Scientific, and Medical (ISM) Band

II. Description of the Dataset

2.1 **Parameter Description**

The Dataset for this problem consists of multi-channel observations of the mixture of two RF signals emitted by the two drones in our scenario:

- A synthetic and noiseless QPSK-modulated signal $\boldsymbol{z_{SOI}}$ emitted by an emulated drone which serves as the signal-of-interest (SOI), and
- A recorded and noisy communication waveform $\boldsymbol{z_{Interferer}}$ transmitted over-the-air by a commercially-available drone which serves as the co-channel interference. The bandwidth of $\boldsymbol{z_{Interferer}}$ is approximately the same as $\boldsymbol{z_{SOI}}$.

Each file consists of a single frame of the SOI combined linearly with the interference, i.e.

$$\boldsymbol{Z_{mixture}} = \boldsymbol{Z_{SOI}} + \alpha\, \boldsymbol{Z_{Interferer}} \quad (1)$$

where α is a scalar coefficient that controls the strength of the interference relative to the SOI. The coefficient α is chosen from a set of 25 possible values selected to produce signal-to-interference-plus-noise (SINR) ratios ranging from -18 to +12 dB. The value of α in effect for a given file in the multisensory Dataset is identified by its positive-integer-valued *alphaIndex*.

The properties of the SOI are listed in Table I below. These parameters are provided for informational purposes only. *Please note: Your methodology should not rely upon knowledge of these specific SOI parameters; rather they should be designed to protect a wide range of structured RF signals beyond this specific SOI.*

Table 1: Parameters for SOI

Parameter	**Value**
Forward Error Correction	Bose-Chaudhuri-Hocquenghem (BCH) (c.f. [1]) Message Bits per Codeword = 24 Codeword Bits per Codeword = 63
Frame Length (Codewords)	Variable number of Codewords *frameLen* = { 4, 6, 8, 10, 12, 16, 32, 64, 128}
Symbol Modulation	QPSK
Samples per symbol	2
Pulse Shaping Filter	Root-raised-cosine with rolloff factor 0.25 Truncated to 12 symbols
Preamble (QPSK) symbols for channel estimation (not included in frame length)	50

2.2 **Filename Convention for Input**

Each file is named according to its *alphaIndex, setIndex,* and frame length *frameLen* (see Table 1 above), and *frame* number, using the following convention:

*input_frameLen_*[frameLen]*_setIndex_*[setIndex]*_alphaIndex_*[alphaIndex]*_frame_*[frameIndex].iqdata

2.3 Dataset Partitioning and Format

The full Dataset is partitioned into sets of files. Each set is associated with a distinct *frameLen*, and consists of 100 distinct frame mixtures for each value of *alphaIndex*. There are 20 sets provided for each value of *frameLen*. You may use any one of these sets to obtain a coarse-level validation and/or performance assessment of their separation algorithm.

Each set is stored as a .zip file with a name of the form:

*rfChallenge_multisensor_frameLen_*[frameLen]*_setIndex_*[setIndex].*zip*

Each individual file within a set, representing a single mixture frame as described above, stores mixture baseband (complex) samples in binary format with 32-bit-float samples from the four channels stored serially and in-phase I (i.e. real) and quadrature Q (i.e. imaginary) components interleaved for each channel. In-phase and quadrature float samples are stored using IEEE lower-endian bit ordering. Reference code written in GNU Octave which loads $\boldsymbol{Z_{mixture}}$ from this file format is provided in the function ***readData_oct.m*** is provided in the *sourceCode* subdirectory of the repo[3].

III. Problem Description

Expanding the terms in Equation (1), the N_{SAMP} samples of a given mixture frame $\boldsymbol{Z_{mixture}}$ captured by the N_R-channel receiver can be modeled as:

$$\boldsymbol{Z_{mixture}} = \boldsymbol{h}_{SOI}\boldsymbol{s} + \boldsymbol{h}_{Interferer}\boldsymbol{b} + \boldsymbol{n} \qquad (2)$$

Where:

- $\boldsymbol{Z_{mixture}}$ is an N_R x N_{SAMP} data matrix containing the received mixture of the SOI and Interferer signals
- $\boldsymbol{s}$ is a 1 x N_{SAMP} vector containing the SOI transmitted signal
- $\boldsymbol{b}$ is a 1 x N_{SAMP} vector containing the Interferer transmitted signal
- $\boldsymbol{h}_{SOI}$ and $\boldsymbol{h}_{Interferer}$ are the N_R x 1 array responses associated with SOI and Interferer, respectively
- $\boldsymbol{n}$ is the N_R x N_{SAMP} vector of receiver noise in the recording of the Interferer

The challenge posed to you is to develop an algorithm which recovers $\hat{\boldsymbol{s}}$, an estimate of $\boldsymbol{s}$ for a given frame using only the observed mixture data $\boldsymbol{Z_{mixture}}$ for that single frame. *Please note that separation approaches which train on multiple frames to recover single-frame estimates are not of interest in this part of RFChallenge*.

[3] Repo URL: https://github.com/RFChallenge/rfchallenge_multichannel_starter

This Problem falls into the general category of *Blind Source Separation*.

IV. Getting Started with Multi-Sensor *RFChallenge*

The instructions below provide setup instructions for evaluating performance of the baseline signal separation method described in Section VI. With a few additional steps described at the end of this Section, these instructions can be extended to perform the same evaluation on your own signal separation algorithm.

4.1. Download and install the free open-source scientific computing package GNU Octave[4] if not already installed on your computing machine of choice

4.2. Download *RFChallenge* Multi-Sensor Dataset available on the MIT-hosted *RFChallenge* website

4.3. Clone RFChallenge Multi-sensor GitHub repo located at the following URL:
https://github.com/RFChallenge/rfchallenge_multichannel_starter
Source code for this part of RFChallenge is located in the repo sub-directory *sourceCode*

4.4. Unzip the .zip file(s) within the downloaded Dataset corresponding to the (frameLen, setIndex) pair(s) on which you want to evaluate separation algorithm performance to a directory of your choosing

4.4.1. Note 1: It is recommended that you test the evaluation process initially with a single .zip file

4.4.2. Note 2: By default, main script expects the directory to be repo sub-directory *mixtureData*

4.5. Specify this directory as a string in the variable *inputDirectory* at the top of main script ***evalMain.m***

4.6. Create a directory for storing the output of the signal separation algorithm

4.6.1. Note: By default, main script will store signal separator output in repo sub-directory *sepOutput*

4.7. Specify this directory in the variable *outputDirectory* at the top of main script ***evalMain.m***

4.8. Unzip the .zip file in the repo *soiParamFiles.zip*. These files, which contain SOI parameters necessary for performance evaluation, should be unzipped to a subdirectory with the name *soiParamFiles*.

4.9. Start GNU Octave

[4] GNUOctave is open-source software and available for download at https://www.gnu.org/software/octave/index

4.10. Install the *communications* package by entering the following in the GNUOctave Command Window: `pkg load communications`

4.11. Find and run the GNU Octave script **evalMain.m** In the repo *sourceCode* sub-directory**.** At this point you should see status printouts in the Octave command window showing the succession of files that are being loaded for processing. For each frame length that is processed, a plot will show the fraction of successfully decoded frames as a function of SINR.

When you have implemented your signal separation algorithm in a programming language of your choice and wish to evaluate its performance, you can do so by following these additional steps:

4.12 Implement interface in your code for reading mixture data from the Dataset files (see Section V for details)

4.13 Implement interface in your code for writing separated signals to files adhering to the naming conventions and file formats described in Section V, so that the quality of the separated output can be evaluated by the GNUOctave script **evalMain**

4.14 Run your signal separation code to generate signal separation output for evaluation

4.15 Comment out the call to the reference algorithm *sigSeparator* in **evalMain**

4.16 Run **evalMain**

4.16.1 Note: Section 5.4 describes how performance is scored

V. Performance Evaluation of Participant Code

5.1 **Evaluation Description**

Performance evaluation will quantify over the full Dataset the frame error rate of the SOI estimates $\hat{\boldsymbol{s}}$ provided by your signal separation code at the receiver. Evaluation code written in GNUOctave is provided for the same in the *RFChallenge* Git repository (see Section 4). Your signal separation code should produce output adhering to the conventions and formatting followed by the Evaluation code, and you should not modify the Evaluation code, except where indicated by comments.

The Evaluation code evaluates the quality of separated signals by executing the following data and signal processing steps in order for each of the frames in a given output file:

1. Basic error-checking on the input

2. Matched filtering on the signal-of-interest (using the root-raised-cosine filter parameters in Table I)
3. Channel estimation using the preamble symbols (c.f. Table I)
4. Equalization using the channel estimates from step 3
5. Demodulation of payload symbols using standard QPSK demodulator
6. Decoding of demodulated symbols using BCH decoder (c.f. Table I for parameters)
7. Comparison of decoded bits with true bits for this frame

Steps 1-7 are repeated for each frame in the file and an overall *frameSuccessRate* is computed as the fraction of frames with zero bit errors. ***This frameSuccessRate metric is the base evaluation parameter on which all Participant solutions will be evaluated (see Section 5.3 "Scoring" for details).***

To successfully run the Evaluation code on the output of your signal separation algorithm, you will need to write estimates of the individual signal components in the mixture to file. ***Please note: For evaluation to be successful, the estimated components written to file should have the same length (in number of samples) as their input mixture data.***

Since Blind Signal Separation methods are by definition not able to associate the components they separate with a particular SOI[5], Your code is *not* required to identify which of the separated components they produce is the SOI. Instead, your code is required to output each separated components to its own file and the provided Evaluation code will check the number of SOI bit errors in each component. If there are zero bit errors in either of the two separated component frames, then decoding is declared a success.

Specifically for each *input* file associated with a distinct (frameLen, setIndex, alphaIndex, frameIndex) tuple as described in Section II, your signal separation code should produce a corresponding single-channel *output* file in the same binary format as the input, *for each of the two separated components*. Reference code written in GNUOctave for writing this single-channel output file is provided in in the function ***writeData_oct.m*** is provided in the *sourceCode* subdirectory of the repo. Note: as the input to this function is single-channel, the data passed to this function to be written to file should be a row vector.

5.2 **Filename Convention for Output (Separated Signal Frames)**

The filename for the two outputs matches the input filename described in 2.2, except with "outputA" or "outputB" replacing "input" at the beginning of the name for the first

[5] This is the so-called *permutation ambiguity* common to blind signal separation algorithms

and second separated signal components, respectively. Namely to be evaluated by **evalMain** script, your code needs to produce the following two output files for tuple (frameLen, setIndex, alphaIndex, frameIndex):

For the first separated component, the filename should be:

*outputA_frameLen_*[frameLen]*_setIndex_*[setIndex]*_alphaIndex_*[alphaIndex]_frame_[frameIndex].iqdata

For the second separated component, the filename should be:

*outputB_frameLen_*[frameLen]*_setIndex_*[setIndex]*_alphaIndex_*[alphaIndex]_frame_[frameIndex].iqdata

5.3 **Running Evaluation Code on Separated Signal Frames**

The GNUOctave script ***evalMain.m*** evaluates the performance of submitted separation algorithms over the full Dataset representing all values of the tuple (frameLen, setIndex, alphaIndex, frameIndex). To evaluate the frame error rate on any individual tuple value, this script calls helper function ***evaluateSeparation.m,*** which takes in the value of the tuple as a comma-separated list and the number of separated signals to process (2), and outputs the fraction of decoded frames with zero bit errors *frameSuccessRate*, and (for purely informational purposes) the bit error rate *ber*. This function, whose prototype is below, can also be used to spot-check performance on a given set of parameters.

```
function [frameSuccessRate, ber] = evaluateSeparation(alphaIndex,
frameLen, setIndex, nSources)
```

5.4 **Scoring**

The final Score used to determine your place on the Multi-Sensor Leaderboard is the minimum value of SINR (over the SINRs instantiated by each *alphaIndex*) at which a frame error rate of less than 10% is achieved. An independent Score is computed for each frame length *frameLen*, and you may place on the Leaderboard for one or more individual values *frameLen*. Smaller scores indicate better separation performance. As exemplified in Figure 3 for the baseline algorithm, lower values of *frameLen* allow the limited levels of sample support that generally challenge unsupervised learning methods (including blind source separation methods). The hope is that you will provide a method which yields stronger interference rejection even at the low-to-moderate frame lengths.

5.5 **Submitting your Solution**

If you wish to be included on the Multi-Sensor Leaderboard on the *RFChallenge* website, please submit a short summary of the multi-sensor signal separation algorithm they

used. The organizers will use this summary to verify that the algorithm is within the guidelines set forth in Section II and III, before including results in the Multi-Sensor RFChallenge Leaderboard. The summary must clearly indicate the minimum-SINR Score reported by running **evalMain** for each attempted frame length (i.e. value of *frameLen*), over all sets (i.e. all 20 values of *setIndex*) and all interference strengths (i.e. all 25 values of *alphaIndex*).

VI. Baseline Method: *Independent Components Analysis*

6.1 Algorithm Description

To provide a performance benchmark for Participant methods, we have selected a well-established blind source separation method known as complex-valued *FastICA* (c.f. [2], [3]) as a baseline method for this part of *RFChallenge*. Applying this method to our problem entails writing the signal mixtures and their components as random variables, yielding a statistical model for the samples of the mixture written in terms of the (scalar) samples of SOI s and interferer b respectively, as follows:

$$\boldsymbol{z_{mixture}} = \boldsymbol{H}\begin{bmatrix} s \\ b \end{bmatrix} \qquad (3)$$

where the matrix $\boldsymbol{H}$ is given by:

$$\boldsymbol{H} = [\boldsymbol{h}_{SOI} \quad \boldsymbol{h}_{Interferer}] \quad (4)$$

Given the model above, the random vector $\widehat{\boldsymbol{x}}$ containing the separated signal components are given by:

$$\widehat{\boldsymbol{x}} = \begin{bmatrix} \hat{s} \\ \hat{b} \end{bmatrix} = \boldsymbol{W^H z_{mixture}} \quad (5)$$

The "unmixing" matrix $\boldsymbol{W}$ has number of columns equal to the number of receive antennas N_R and the number of rows equal to the number of independent signals *nSignals* (in this case 2). Note that in this context the matrix $\boldsymbol{W}$ is in fact a *beamforming matrix* with each column $\boldsymbol{w_j}$ providing the beamforming weights associated with a given signal component. It is recovered in *FastICA* by solving the following optimization problem [2]:

$$\textit{maximize} \sum_{j=1}^{n} J_G(\boldsymbol{w_j}) \text{ with respect to } \{\boldsymbol{w_j}\}\text{, j = 1,..., } \textit{nSignals} \qquad (6)$$

under the constraint:

$$E\{(w_k^H x)(w_k^H x)^*\} = \delta_{jk}$$

Where $\delta_{jk} = 1$ for *j = k*, and $\delta_{jk} = 0$ otherwise

The solution of this optimization problem provided in [2] is an iterative algorithm which solves for the beamforming weight vector $\boldsymbol{w_j}$ associated with each signal (or *independent component)* sequentially. While for brevity we omit the details of this iterative update here, the algorithm is implemented in the file ***complexFastICA.m*** in the *sourceCode* sub-directory of the Multi-Sensor *RFChallenge* Git repository described in Section IV.

In the remainder of this section we describe the objective (or "contrast") function and constraint in (6), and ambiguities involved in this optimization.

6.1.1 Contrast Function and Constraint

The objective or "contrast" function $J(.)$ is of the form:

$$J_G(\boldsymbol{w}) = \boldsymbol{E}\{\boldsymbol{G}(|\boldsymbol{w^H x}|^{\boldsymbol{2}})\} \quad (7)$$

and is typically a metric that quantifies non-Gaussianity (e.g. kurtosis) so that the optimization (6) is designed to produce outputs which are highly non-Gaussian. The function $\boldsymbol{G}(\boldsymbol{y})$ is a smooth even function. If one chooses $(y) = y^2$, then from (6) our contrast function becomes:

$$J_G(\boldsymbol{w}) = \boldsymbol{E}\{|\boldsymbol{w^H x}|^{\boldsymbol{4}}\} \qquad (8)$$

which is basically a measure of the kurtosis of $\boldsymbol{w^H x}$.

In practice kurtosis-based contrast functions can be sensitive to outliers in the data. Hence in practice more robust functions which increase more gradually with their argument are chosen for $\boldsymbol{G}$. For our baseline implementation, we choose:

$$\boldsymbol{G}(\boldsymbol{y}) = \boldsymbol{log}(\boldsymbol{c} + \boldsymbol{y}) \qquad (9)$$

where $\boldsymbol{c}$ is a small constant ($\boldsymbol{c} = \boldsymbol{0.1}$ was chosen for our implementation). Interested Participants are referred to references [2] and [3] for further detail on theory-driven choice of contrast functions for maximizing non-Gaussianity.

Given that combining independent non-Gaussian signals tends to increase Gaussianity relative to that of the individual components, the optimization in (6) will typically favor beamforming weights $\boldsymbol{w_{SOI}}$ which null out the interferer contributions in estimating the

SOI signal *s*. Underlying this qualitative argument is the assumption that the individual components themselves are non-Gaussian, as most man-made RF signals are.

The role of the decorrelation constraint $E\{(w_k^H x)(w_j^H x)^*\} = \delta_{jk}$ is to enforce that the procedure finds *distinct* independent signal components, as opposed to producing repeated beamforming weights which select the same component.

6.1.2 Ambiguities

Like most blind source separation methods, this method solves for the independent signal components in a mixture up to permutation and scaling ambiguities. Namely the target solution of optimization (6), i.e. the separated components vector $\boldsymbol{x}_{target}$, satisfies:

$$\boldsymbol{x}_{target} = \boldsymbol{PD}\begin{bmatrix} s \\ b \end{bmatrix} \qquad (10)$$

where $\boldsymbol{P}$ is a permutation matrix taking values in {0, 1}, and $\boldsymbol{D}$ is a diagonal scaling matrix whose diagonal elements are arbitrary complex scalars.

6.2 Performance Evaluation on Dataset

The FastICA algorithm described in Section 6.1 was applied on a frame-by-frame basis to obtain a beamforming weight matrix $\boldsymbol{W}$ for each frame. The N_{SAMP} samples of the separated signal components $\{\hat{\boldsymbol{s}}_k, \widehat{\boldsymbol{b}}_{\boldsymbol{k}}\}$ in the *k-th* frame were then obtained by applying the beamforming matrix $\boldsymbol{W}_{\boldsymbol{k}}$ to the N_R x N_{SAMP} data matrix $\boldsymbol{Z}_{\boldsymbol{mixture}}$ (described in Section III) of the k-th frame as follows:

$$\begin{bmatrix} \hat{\boldsymbol{s}}_k \\ \widehat{\boldsymbol{b}}_{\boldsymbol{k}} \end{bmatrix} = \boldsymbol{W}_{\boldsymbol{k}}^{\boldsymbol{H}} \boldsymbol{Z}_{\boldsymbol{mixture}}(\boldsymbol{k})$$

Due to the permutation ambiguity described in Section 6.1.2, both outputs were presented to the Evaluation algorithm as distinct files prefixed with “outputA” and “outputB” respectively (see Section 5.1 and 5.2), for every frame in the Dataset.

6.3 Results

The results for a particular frame length (*frameLen* = 32 words) are shown in Figure 2.

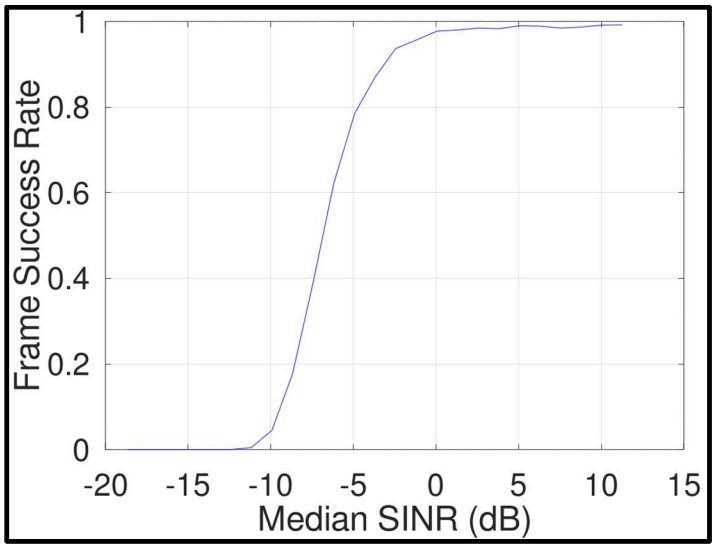

Figure 2: FastICA performance for 32-word frame length across full Dataset

The minimum SINR required to achieve a frame success rate of 90% (i.e. the scoring function described in Section 5.4) is plotted in Figure 3 as a function of frame length in codewords (bottom x-axis) and symbols (top x-axis). For comparison, decoding a single channel of the mixture frames with no interference mitigation requires an SINR of ~5 dB for a frame success rate of 90%. As a benchmark for performance, we also assessed the performance of an MMSE estimator with oracle knowledge of the array responses of the SOI $\boldsymbol{h}_{SOI}$ as well as the interference-plus-noise spatial covariance matrix $\boldsymbol{R}_{ni}$. The MMSE beamformer for the SOI is then constructed as (c.f. [4]):

$$\boldsymbol{w_{SOI}} = \boldsymbol{R}_{ni}{}^{-1}\boldsymbol{h}_{SOI}$$

This *oracle* MMSE beamformer provided 90% frame success rate at SINRs above -9 dB, as shown in Figure 3.

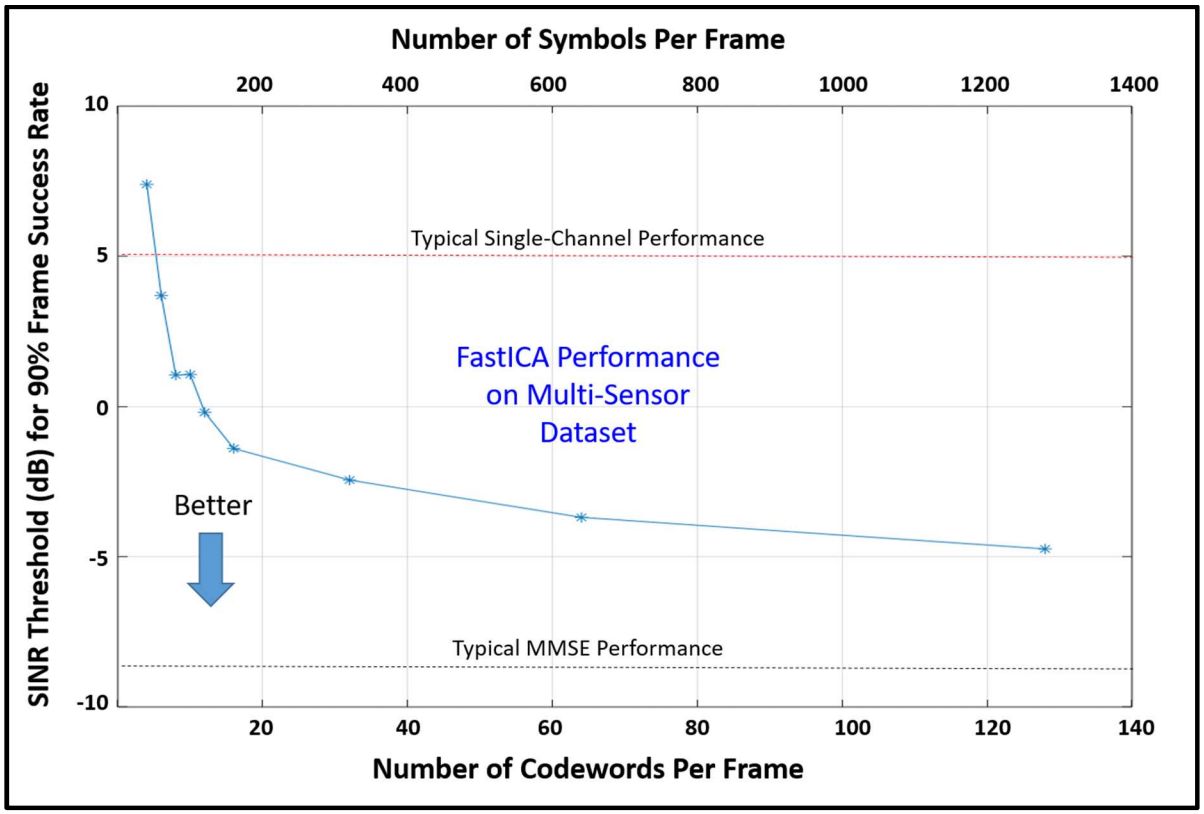

Figure 3: FastICA Achieved Score as a function of *frameLen*

VII. References

[1] B. Sklar. *Digital Communications: Fundamentals and Applications*, 2nd Edition, Upper Saddle River: Prentice Hall PTR, 2000, pp. 370-374

[2] E. Bingham and A. Hyvarinen. A fast fixed-point algorithm for independent component analysis of complex-valued signals. *Int. J. of Neural Systems,* 10(1):1-8,2000

[3] A. Hyvarinen, J. Karhunen, E. Oja. *Independent Component Analysis*. New York: John Wiley & Sons, 2001.

[4] D. Tse and P. Viswanath. *Fundamentals of Wireless Communication*. New York: Cambridge University Press, 2005.