Towards Real-Time High-Resolution Interferometric Imaging with Bluebild

Master Thesis, Spring 2017

SEPAND KASHANI, École Polytechnique Fédérale de Lausanne (EPFL) and IBM Research – Zurich
DR. PAUL HURLEY, Supervisor, IBM Research – Zurich
DR. ADAM SCHOLEFIELD, Supervisor, LCAV, École Polytechnique Fédérale de Lausanne (EPFL)

Rooted in radio-astronomy, the Bluebild [16] algorithm is a paradigm-changing imager to estimate the sky intensity function. However, current imaging pipelines in radio-astronomy are highly-tailored to the CLEAN-family of imagers, and hence lack the flexibility required to properly experiment with Bluebild and its associated technologies. Therefore, there is a need to design tools flexible enough to allow such algorithms to be implemented while retaining a large degree of freedom for future experimentation.

Owing to its low computational complexity and affinity for parallel execution, Bluebild is a good candidate to achieve real-time imaging in the field. Moreover, its generality makes it broadly applicable to other domains as well, such as sound-field imaging. The goal of this thesis is therefore two-fold: First, we tackle the system aspects of Bluebild, working towards the goal of doing real-time imaging in radio-astronomy and acoustics. This is done by looking at the problem from an algorithm and architecture point-of-view. Secondly, we design *Pypeline*, a modular Python toolkit to solve the infrastructure crisis Bluebild faces at the moment.

For radio astronomy, we leveraged algorithm speed-ups and system optimizations to reduce computation by an order of magnitude on CPU architectures, and showed that Bluebild can be efficiently implemented on GPUs by exploiting the inherent parallelism of the sampling stage. We additionally showed the first least-square image estimates using wide-angle beamformers.

Based on the radio astronomy Bluebild implementation, we built a real-time sound field imager on top of the Pyramic microphone array that accounts for the specifics of acoustic arrays. We showed by experiment that reconstructions of Bluebild seem to be at least on par with those of B-scan and MUSIC, while being more stable than MVDR.

Finally, we developed *Pypeline*, a Python package to design composable signal processing systems from elementary building blocks. Through its flexible design, it allowed us to implement entire imaging pipelines for many sensor arrays, allowing technologies such as Flexibeam and randomized beamforming to be tested for the first time with Bluebild.

Contents

1 INTRODUCTION

In radio-astronomy, interferometry is a key investigative technique to form estimates of the sky intensity function $I(r)$. For the past 30 years, the CLEAN [9] family of imagers have been the backbone of imaging pipelines in the field to estimate this quantity. Although tractable with current telescope designs, it is already known that the state-of-the-art imaging pipeline cannot scale to the requirements set by next-generation designs such as the SKA [8]. As such, there is a need to develop novel imaging algorithms to make future telescope designs usable.

Rooted in radio-astronomy, the Bluebild [16] algorithm is a paradigm-changing imager to estimate $I(r)$. Based on the theory of sampling and interpolation operators [19], Bluebild forms an L_2-estimate of $I(r)$ through a continuously-defined left pseudo-inverse to the instrument's sampling operator. In addition to its low computational complexity, Bluebild enjoys many appealing properties:

Model correctness: sky estimates $I(r)$ are defined on the sphere, making the reconstruction more faithful to the true nature of the sky as perceived by radio-telescopes.

Numerical stability: solving large-scale inverse problems is the root cause for ill-conditioning in current imaging pipelines. Bluebild however does not suffer from this problem as it only needs to solve a small generalized eigenvalue problem of the form $\Sigma x = \lambda G x$, for which numerically stable and efficient solvers exist.

Flexibility: arbitrary beamforming strategies are easily incorporated in the imaging problem by modifying the instrument's sampling and interpolation operators.

Computational efficiency: since $I(r)$ is continuously-defined, it can be sampled at arbitrary resolutions for visualization.

Given the above, it is foreseen that Bluebild will scale past the limitations of current imaging pipelines.

At the same time, Bluebild's generality makes it broadly applicable to other domains such as acoustics. Although never tried until now, the algorithm can be used in conjunction with a microphone array to form an image of the sound field around the instrument. This estimate can then be used to locate sources and track their movement through time.

For such functionality to provide meaningful results however, we need the ability to run Bluebild in real-time. In acoustics, this amounts to forming snapshots of the sound field at a moderate frame-rate to allow precise tracking. Similarly, real-time computing also has its importance in radio-astronomy, as the SKA will have a so-called "fast-imaging" mode. Concretely, for a given hardware budget and image quality, fast-imaging amounts to producing sky snapshots of multiple subbands between imaging cycles. Current imaging pipelines already cannot compete in this space.

To use Bluebild and evaluate its performance, it is necessary to incorporate it as a module in an imaging pipeline. However, current tools in radio-astronomy are too tailored to CLEAN [9] and hence cannot take advantage of all Bluebild's features such as generalized beamforming. In addition, as Bluebild can be used across domains, we would like an imaging pipeline that favors generality over domain-specific solutions. No such tool exists at this time that offers the level of flexibility we desire. Thus, there is a need to develop a toolkit that is flexible enough to offer room for experimentation with Bluebild, and has wide applicability across domains. Such a system can also serve as an educational tool, to teach people about Bluebild.

The goal of this thesis is therefore two-fold: First, we tackle the system aspects of Bluebild, working towards the goal of doing real-time imaging in radio-astronomy and acoustics. This is done by looking at the problem from an algorithm and architecture point-of-view. Secondly, we design *Pypeline*, a modular Python toolkit to solve the infrastructure crisis Bluebild faces at the moment.

To summarize, this thesis is decomposed as follows:

- In Section 2, we set the data model and formulate Bluebild using the mathematical notion of sampling/interpolation operators. Using a sample application in acoustic imaging, we look at the various properties of the algorithm. Moreover, we compare the Bluebild reconstruction to well-known imaging techniques in acoustics.
- Section 3 is composed of two parts: First, we focus on acoustics and develop optimizations specific to this context to achieve real-time imaging of the sound field. Moreover, we physically build a real-time sound-field imager using a microphone array. We then change focus to radio-astronomy. Due to the rotation of the Earth, optimizations formerly applicable in acoustics no longer hold here, hence we explore alternative acceleration schemes that take into account limitations of modern radio-telescopes and inherent parallelism of Bluebild.
- Section 4 presents *Pypeline*, our Python package to design composable signal processing systems from elementary building blocks. After a brief overview of the main concepts of Pypeline, we walk through an example in radio-astronomy to show how the tool's flexibility allows rapid experimentation and prototyping with Bluebild.
- Section 5 concludes this work by summarizing our contributions and setting the stage for future research on Bluebild.

2 IMAGING FUNDAMENTALS

Many real-life objects of interest can be modelled by a random process $\mathcal{S}$. In radio-astronomy for example, the sky emissions are modelled as follows [1, 15, 16, 18]:

Definition 2.1 (Far-field sky model). Let $(\Omega, \mathcal{F}, \mathbb{P})$ be some probability space. Source emissions are modelled as realizations of a stationary narrow-band Gaussian random field

$$\mathcal{S} = \left\{ S(r) : \Omega \to \mathbb{C},\ r \in \mathbb{S}^2 \right\},$$

where $\mathbb{S}^2$ denotes the sphere and $S(r)$ can be written in baseband-equivalent form as

$$S(r) = \widehat{S}(r) e^{j2\pi f_c t}, \quad \widehat{S}(r) \sim \mathcal{N}_c\left(0, I(r)\right), \tag{1}$$

with $\mathcal{N}_c$ being the complex Gaussian distribution. Moreover, signals coming from different directions in the sky are uncorrelated:

$$\mathbb{E}\left[S(r_1) S^*(r_2)\right] = 0, \quad \forall (r_1, r_2) \in \mathbb{S}^2.$$

Similarly, when acoustic sources can be stationary and located in the far-field, the same data model holds.

Notice that the distribution of $\mathcal{S}$ depends only on the sky intensity function $I_{\mathcal{S}}(r) = \mathbb{E}\left[S(r)S^*(r)\right]$, hence knowledge of this function is sufficient[1] to fully characterize the field. Estimating $I_{\mathcal{S}}$ requires gathering information on $\mathcal{S}$ through observation using an acquisition device. However instruments are usually not transparent: their effect on the observed realizations must be modelled and taken into account in estimating $\mathcal{S}$. Sticking to radio-astronomy and acoustics, realizations of $\mathcal{S}$ can be collected with a radio-telescope or a microphone-array [15]:

Definition 2.2 (Radio-telescope / Microphone-array model). Consider L antennas[2] located at positions $p_1, \ldots, p_L \in \mathbb{R}^3$. Assuming far-field sky model 2.1, samples at antenna l take the form

$$X_l = \int_{\mathbb{S}^2} S(r)\alpha_l^*(r)e^{-j\frac{2\pi}{\lambda_c}\langle p_l, r\rangle}dr, \quad l \in \{1, \ldots, L\}, \tag{2}$$

where $\alpha_l : \mathbb{S}^2 \to \mathbb{C}$ is the antenna gain function modelling its sensitivity to signals coming from different directions, λ_c is the wavelength of incoming signals, and $e^{-j\frac{2\pi}{\lambda_c}\langle p_l, r\rangle}$ is a phase term that accounts for the signal's time-delay between different antennas. Due to the high data-rates and large number of antennas, samples $X \in \mathbb{C}^L$ are often not available directly, but beamformed together using a weighting matrix $W \in \mathbb{C}^{L\times M}$ to achieve compression:

$$Y = W^H X.$$

Given the transformations done on $\mathcal{S}$ by the instrument, how can one estimate $I_{\mathcal{S}}$ by only having access to $Y \in \mathbb{C}^M$? The theory of sampling and interpolation operators [19] provides the answer. Hence, we start off by a brief overview of these concepts in the context of far-field signals shown above, before diving into the Bluebild algorithm to estimate $I_{\mathcal{S}}$.

2.1 Sampling & Interpolation Operators

Assume $\mathcal{S} : \mathbb{S}^2 \to \mathbb{C}$ is a zero-mean random field whose covariance function $\kappa_{\mathcal{S}}(r_1, r_2)$ is given by

$$\begin{aligned}\kappa_{\mathcal{S}} : \mathbb{S}^2 \times \mathbb{S}^2 &\to \mathbb{C}\\ (r_1, r_2) &\to \mathbb{E}\left[S(r_1)S^*(r_2)\right].\end{aligned}$$

Estimating $\kappa_{\mathcal{S}}$ requires observing realizations of $\mathcal{S}$ using an acquisition device. The acquisition device is not transparent: its effect can be viewed as a sampling operator Ψ^* composed of 3 operations [16] (Figure 2a):

- Spatial filtering of $\mathcal{S}$ by a device-dependent kernel $\phi : \mathbb{S}^2 \times \mathbb{R}^3 \to \mathbb{C}$, obtaining measurement field $\mathcal{X} : \mathbb{R}^3 \to \mathbb{C}$:

$$\mathcal{X} = \left\{\widetilde{X}(p) : \Omega \to \mathbb{C},\ p \in \mathbb{R}^3\right\},$$

$$\widetilde{X}(p) = \langle \mathcal{S}, \phi(\cdot, p)\rangle_{\mathbb{S}^2} = \int_{\mathbb{S}^2} S(r)\phi^*(r, p)dr. \tag{3}$$

- Spatial sampling of $\mathcal{X}$ at specific locations $p_1, \ldots, p_L \in \mathbb{R}^3$:

$$X = \left[\widetilde{X}(p_1), \ldots, \widetilde{X}(p_L)\right] \in \mathbb{C}^L. \tag{4}$$

- Beamforming of acquired samples using a full-rank linear operator $W : \mathbb{C}^M \to \mathbb{C}^L$, where $M \leq L$:

$$Y = W^H X \in \mathbb{C}^M.$$

By defining $\phi_l(r) = \phi(r, p_l),\ l \in \{1, \ldots, L\}$ and $\psi_m(r) = \sum_{l=1}^{L} W_{lm}\phi_l(r),\ m \in \{1, \ldots, M\}$, the compound effect of Ψ^* on $\mathcal{S}$ can conveniently be written

$$\begin{aligned} Y = \Psi^*\mathcal{S} = W^H\Phi^*\mathcal{S}, &\qquad \text{where}\\ \Phi^*\mathcal{S} = \left[\langle \mathcal{S}, \phi_1\rangle_{\mathbb{S}^2}, \ldots, \langle \mathcal{S}, \phi_L\rangle_{\mathbb{S}^2}\right], &\\ \Psi^*\mathcal{S} = \left[\langle \mathcal{S}, \psi_1\rangle_{\mathbb{S}^2}, \ldots, \langle \mathcal{S}, \psi_M\rangle_{\mathbb{S}^2}\right]. &\end{aligned} \tag{5}$$

Interpolating Y back onto $\mathbb{S}^2$ using a suitable interpolation operator $\widetilde{\Psi} : \mathbb{C}^M \to \mathbb{S}^2$ gives us

$$\widetilde{\mathcal{S}} = \left\{\widetilde{S}(r) : \Omega \to \mathbb{C},\ r \in \mathbb{S}^2\right\},$$

$$\widetilde{S}(r) = \sum_{m=1}^{M} \widetilde{\psi}_m(r)Y_m.$$

The covariance function can then be reconstructed as a weighted sum of separable kernels:

$$\begin{aligned}\kappa_{\widetilde{\mathcal{S}}}(r_1, r_2) &= \mathbb{E}\left[\widetilde{S}(r_1)\widetilde{S}^*(r_2)\right]\\ &= \sum_{i,j=1}^{M} \widetilde{\psi}_i(r_1)\,\mathbb{E}\left[Y_iY_j^*\right]\widetilde{\psi}_j^*(r_2)\\ &= \sum_{i,j=1}^{M} \Sigma_{ij}\widetilde{\psi}_i(r_1)\widetilde{\psi}_j^*(r_2), \qquad \forall (r_1, r_2) \in \mathbb{S}^2.\end{aligned} \tag{6}$$

As we want the estimate $\kappa_{\widetilde{\mathcal{S}}}$ to faithfully represent $\kappa_{\mathcal{S}}$, $\widetilde{\Psi}$ should satisfy some sort of optimality criteria. The least-squares solution is obtained when the pair $(\widetilde{\Psi}, \Psi)$ is consistent and ideally-matched [19], which is guaranteed when $\widetilde{\Psi}$ is chosen to be the pseudo-inverse of Ψ^*:

$$\widetilde{\Psi} = \Psi G_{\Psi}^{-1} = \Psi(\Psi^*\Psi)^{-1}. \tag{7}$$

The matrix $G_\Psi \in \mathbb{C}^{M\times M}$ is called the *Gram matrix* of the instrument and is defined as

$$\begin{aligned}(G_\Psi)_{ij} &= \langle \psi_i, \psi_j\rangle_{\mathbb{S}^2}\\ &= \int_{\mathbb{S}^2} \psi_i(r)\psi_j^*(r)dr, \qquad \forall (i, j) \in \{1, \ldots, M\}.\end{aligned}$$

It's role is to correct acquired samples Y for the non-orthogonality of the instrument basis functions $\{\psi_m,\ m \in \{1, \ldots, M\}\}$[3]. Combining (6) and (7), it can be shown [16] that

$$\kappa_{\widetilde{\mathcal{S}}}(r_1, r_2) = \sum_{i,j=1}^{M} \widetilde{\Sigma}_{ij}\psi_i(r_1)\psi_j^*(r_2), \quad \forall (r_1, r_2) \in \mathbb{S}^2, \tag{8}$$

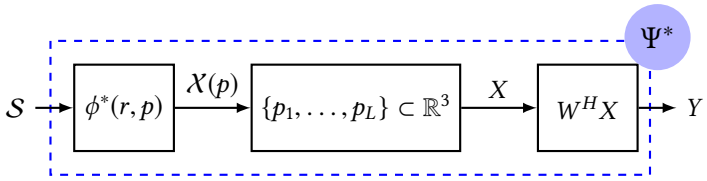
$$I_{\widetilde{\mathcal{S}}}(r) = \kappa_{\widetilde{\mathcal{S}}}(r, r) = \sum_{i,j=1}^{M} \widetilde{\Sigma}_{ij}\psi_i(r)\psi_j^*(r), \qquad \forall r \in \mathbb{S}^2, \tag{9}$$

where $\widetilde{\Sigma} = G_\Psi^{-1}\Sigma G_\Psi^{-1}$.

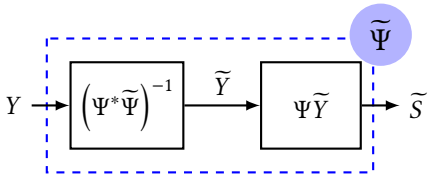
[1] $I_{\mathcal{S}}(r)$ is a sufficient statistic because we assumed that sources coming from different directions are uncorrelated. If this is not the case, the sufficient statistic for a zero-mean Gaussian random field would be $\kappa_{\mathcal{S}}(r_1, r_2) = \mathbb{E}\left[S(r_1)S^*(r_2)\right]$.

[2] "antenna" and "microphone" will be used interchangeably depending on the context.

[3] Indeed, if the basis functions are mutually orthogonal, then $G_\Psi = I_M$.



(a) Sampling operator



(b) Ideally-matched interpolation operator

Fig. 2. Schematic view of sampling and ideally-matched interpolation operators. Figures based on [16].

2.2 Bluebild

Computing $\kappa_{\widetilde{\mathcal{S}}}, I_{\widetilde{\mathcal{S}}}$ with (8) and (9) has several drawbacks. On the one hand, it requires the evaluation of all M^2 kernels $\psi_i(r_1)\psi_j^*(r_2)$, which can be significant for large instruments. This can be mitigated by making the beamforming stage a compressive operator, i.e. having $M < L$, but this lossy operation comes at the cost of reconstruction quality. On the other hand, we need to explicitly evaluate G_Ψ^{-1} to compute $\widetilde{\Sigma}$. This is undesirable as the Gram matrix can be ill-conditioned if the instrument basis functions $\{\psi_m,\ m \in \{1,\dots,M\}\}$ are strongly correlated.

Bluebild [16] solves both problems by expressing $\kappa_{\widetilde{\mathcal{S}}}$ in a compact orthogonal basis $\mathcal{B}$ using a functional PCA decomposition. Concretely, by finding the M eigenpairs (λ_m, υ_m) solution of the generalized eigenvalue problem $\Sigma\upsilon = \lambda G_\Psi \upsilon$, and defining the normalized eigenfunctions

$$\epsilon_m = \frac{\Psi\upsilon_m}{\|\Psi\upsilon_m\|} = \frac{\sum_{k=1}^M \psi_k \upsilon_{km}}{\sqrt{\upsilon_m^H G_\Psi \upsilon_m}}, \qquad m \in \{1,\dots,M\},$$

the covariance function $\kappa_{\widetilde{\mathcal{S}}}$ and intensity function $I_{\widetilde{\mathcal{S}}}$ can be recast [16] as

$$\kappa_{\widetilde{\mathcal{S}}}(r_1,r_2) = \sum_{m=1}^M \lambda_m \epsilon_m(r_1)\epsilon_m^*(r_2), \quad \forall (r_1,r_2) \in \mathbb{S}^2, \tag{10}$$

$$I_{\widetilde{\mathcal{S}}}(r) = \kappa_{\widetilde{\mathcal{S}}}(r,r) = \sum_{m=1}^M \lambda_m |\epsilon_m(r)|^2, \qquad \forall r \in \mathbb{S}^2. \tag{11}$$

In addition to the numerical stability and computational efficiency of Bluebild shown above, computing $\kappa_{\widetilde{\mathcal{S}}}$ and $I_{\widetilde{\mathcal{S}}}$ with Bluebild has other advantages as well:

Accuracy / efficiency trade-off Using (10) and (11) provides exact results for $\kappa_{\widetilde{\mathcal{S}}}$ and $I_{\widetilde{\mathcal{S}}}$, but depending on the decay profile of the eigenvalues $\{\lambda_m,\ m \in \{1,\dots,M\}\}$, it is possible to estimate $\kappa_{\widetilde{\mathcal{S}}}$ and $I_{\widetilde{\mathcal{S}}}$ with $K \ll M$ eigenfunctions.

$$\kappa_{\widetilde{\mathcal{S}}}(r_1,r_2) = \sum_{m=1}^K \lambda_m \epsilon_m(r_1)\epsilon_m^*(r_2),$$

$$I_{\widetilde{\mathcal{S}}}(r) = \sum_{m=1}^K \lambda_m |\epsilon_m(r)|^2,$$

thus achieving significant computational savings. By setting a threshold $\tau \in [0,1]$, we can therefore tune the reconstruction quality with ease. Moreover, small eigenvalues are typically associated with signals having the least energy and/or noise, hence truncating the spectrum can implicitly denoise the estimate.

Energy-specific post-processing In radio-astronomy, it is very common to post-process $I_{\widetilde{\mathcal{S}}}$ using deconvolution algorithms such as CLEAN [9]. By decomposing $I_{\widetilde{\mathcal{S}}}$ into distinct energy-levels, Bluebild allows one to tailor the post-processing to each energy-level to obtain better estimates. For example, we can artificially boost the intensity of weak sources by setting $\lambda_m = 1,\ \forall m \in \{1,\dots,M\}$ in (11).

To summarize, the generic Bluebild algorithm is shown in Algorithm 1.

Algorithm 1: Bluebild(Y, Ψ, τ)

Input: $Y \in \mathbb{C}^{M\times N_s}$, $\Psi : \mathbb{C}^M \to \mathbb{S}^2$, $\tau \in [0,1]$
Output: $\kappa : \mathbb{S}^2 \times \mathbb{S}^2 \to \mathbb{C}$, $I : \mathbb{S}^2 \to \mathbb{R}$

1 Estimate covariance matrix $\Sigma \in \mathbb{C}^{M\times M}$:

$$\widehat{\Sigma} \leftarrow \frac{1}{N_S} Y Y^H$$

2 Compute Gram matrix $G_\Psi \in \mathbb{C}^{M\times M}$:

$$G_\Psi \leftarrow \Psi^*\Psi$$

3 Find M eigenpairs (λ_m, υ_m) of the generalized eigenvalue problem

$$\widehat{\Sigma}\upsilon_m = \lambda_m G_\Psi \upsilon_m, \qquad \forall m \in \{1,\dots,M\}$$

4 Select K leading eigenpairs (λ_m, υ_m) such that

$$\tau \le \sum_{m=1}^K \lambda_m \Big/ \sum_{m=1}^M \lambda_m$$

5 Evaluate eigenfunctions $\epsilon_m : \mathbb{S}^2 \to \mathbb{C}$:

$$\epsilon_m(r) = \frac{\sum_{k=1}^M \psi_k(r)\upsilon_{km}}{\sqrt{\upsilon_m^H G_\Psi \upsilon_m}}, \quad m \in \{1,\dots,K\}$$

6 Form $\kappa(r_1,r_2)$ and $I(r)$:

$$\kappa(r_1,r_2) = \sum_{m=1}^K \lambda_m \epsilon_m(r_1)\epsilon_m^*(r_2), \quad \forall (r_1,r_2) \in \mathbb{S}^2$$

$$I(r) = \sum_{m=1}^K \lambda_m |\epsilon_m(r)|^2, \qquad \forall r \in \mathbb{S}^2$$

2.3 Case Study: Sound Field Imaging with Bluebild

To illustrate the imaging capabilities of Bluebild, let us consider the scenario of Figure 3: an intelligent robot walking around

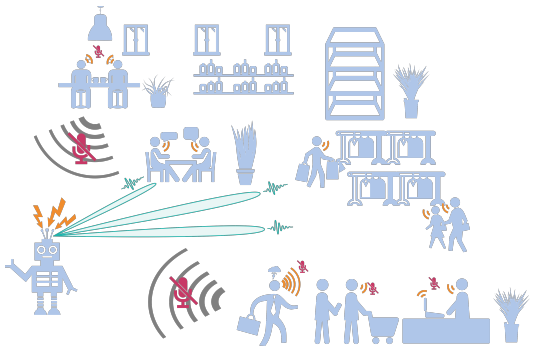
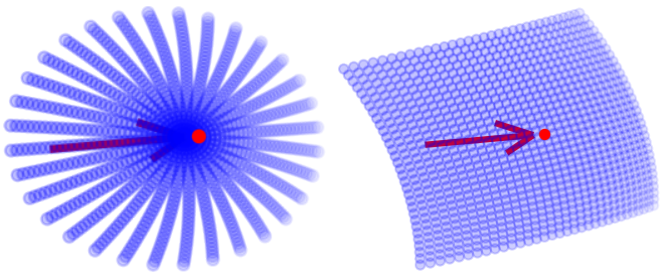


Fig. 3. Microphone-equipped robot roaming a busy train station. Image courtesy of Matthieu Simeoni.



(a) Spherical grid (b) Uniform grid

Fig. 4. Different grid geometries on which to sample Ψ. The figure shows 1024-point grids centered at $\varphi = \theta = \frac{\pi}{4}$ in spherical coordinates, spanning a 60° field-of-view (FoV).

a busy train station. Its purpose in life is to freely roam the station, approach travelers in need of help, and guide them on their journey. To do so, the robot needs to first locate the troubled user in the station and walk in their direction. It is therefore equipped with a microphone array, and can process the recorded audio samples to estimate the user's location w.r.t. its current position.

Mathematically, the robot's dilemma can be formulated as follows: travelers are modelled as a random field $\mathcal{S}$ composed of Q point-sources located in the far-field w.r.t. the robot, and emit stationary narrow-band voice signals as defined in far-field sky model 2.1. In other words, $S(r) = \widehat{S}(r)e^{j2\pi f_c t}$, where

$$\widehat{S}(r) = \sum_{q=1}^{Q} \xi_q \delta(r - r_q), \qquad \xi_q \sim \mathcal{N}_c\left(0, \sigma_q^2\right). \tag{12}$$

User locations are revealed by peaks in the intensity map $I_{\widetilde{\mathcal{S}}}(r)$ using Bluebild. Audio samples are acquired using an L-element microphone array whose effect on $\mathcal{S}$ is given by instrument model 2.2. From (2), (3) and (4), we derive the filtering kernels $\{\phi_l,\ l \in \{1, \ldots, L\}\}$:

$$\begin{aligned} \phi_l : \mathbb{S}^2 &\to \mathbb{C} \\ r &\to \alpha_l(r) e^{j\frac{2\pi}{\lambda_c}\langle p_l, r\rangle}. \end{aligned}$$

Without loss of generality, we assume all microphones are omni-directional, i.e. $\alpha_l(r) = 1,\ \forall (l, r) \in \{1, \ldots, L\} \times \mathbb{S}^2$. The Gram matrix of the instrument is

$$G_\Psi = \Psi^*\Psi = W^H \Phi^* \Phi W = W^H G_\Phi W,$$

where an analytical form exists for G_Φ [16, 17]:

$$\begin{aligned} (G_\Phi)_{ij} &= \langle \phi_i, \phi_j \rangle_{\mathbb{S}^2} \\ &= \int_{\mathbb{S}^2} \phi_i(r) \phi_j^*(r) dr \\ &= \int_{\mathbb{S}^2} e^{j\frac{2\pi}{\lambda_c}\langle p_i - p_j, r\rangle} dr \\ &= 4\pi \operatorname{sinc}\left(2 \left\| \frac{p_i - p_j}{\lambda_c} \right\|\right). \end{aligned} \tag{13}$$

Finally, for visualization purposes, we need to define a grid of sky-points $r \in \mathbb{S}^2$ on which to evaluate the synthesis kernels $\{\psi_m,\ m \in \{1, \ldots, M\}\}$. Bluebild poses no restrictions on the grid-points, hence non-uniform sampling grids can be used. Nevertheless, spherical and uniform grids as shown in Figure 4 are common. Putting in all together, Algorithm 2 shows a concrete procedure to get $I_{\widetilde{\mathcal{S}}}$ using Bluebild. Sample reconstructions of $I_{\widetilde{\mathcal{S}}}$ are shown in Figure 5, both for wide and narrow field-of-views. We notice that peaks in $I_{\widetilde{\mathcal{S}}}$ correspond to the positions of actual sources (black circles).

Algorithm 2: Bluebild(Y, P, W, τ, λ_c, x_{grid})

Input: $Y \in \mathbb{C}^{M \times N_s}$: instrument samples
Input: $P \in \mathbb{R}^{L \times 3}$: antenna positions
Input: $W \in \mathbb{C}^{L \times M}$: beamforming weights
Input: $\tau \in [0, 1]$: energy threshold
Input: $\lambda_c \in \mathbb{R}$: source signal wavelength
Input: $x_{\text{grid}} \in \mathbb{R}^{N_{\text{px}} \times 3}$: grid-points in Cartesian coordinates
Output: $I_{\widetilde{\mathcal{S}}} \in \mathbb{R}^{N_{\text{px}}}$: intensity function

$\widehat{\Sigma} \leftarrow \frac{1}{N_s} Y Y^H$ // $\widehat{\Sigma} \in \mathbb{C}^{M \times M}$
$G_\Psi \leftarrow W^H \Phi^* \Phi W$, with $\Phi^*\Phi$ as in (13) // $G_\Psi \in \mathbb{C}^{M \times M}$
Find M eigenpairs (λ_m, v_m) of the generalized eigenvalue problem

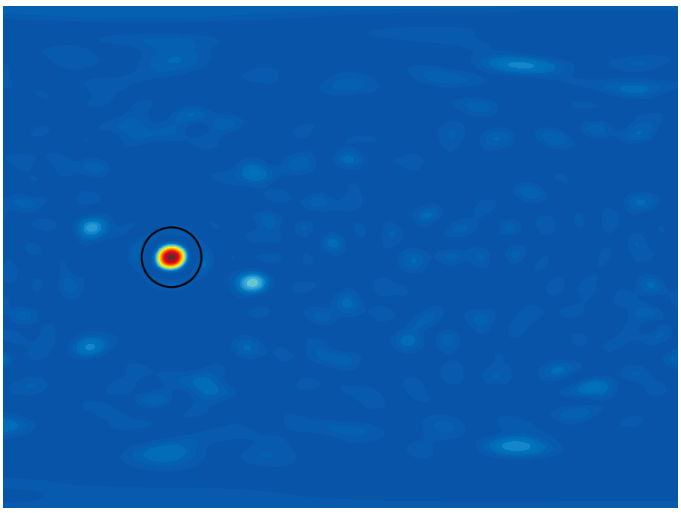
$$\widehat{\Sigma} v_m = \lambda_m G_\Psi v_m, \qquad \forall m = \{1, \ldots, M\}$$

Select K leading eigenpairs (λ_m, v_m) such that

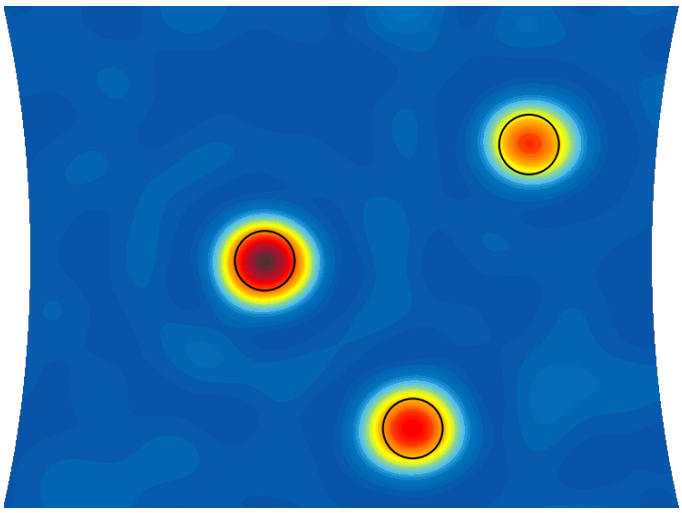
$$\tau \leq \sum_{m=1}^{K} \lambda_m \Big/ \sum_{m=1}^{M} \lambda_m$$

for $m = \{1, \ldots, K\}$ **do**
  $v_m \leftarrow v_m \Big/ \sqrt{v_m^H G_\Psi v_m}$
end
$\Phi \leftarrow e^{j\frac{2\pi}{\lambda_c} x_{\text{grid}} P^T}$ // $\Phi \in \mathbb{C}^{N_{\text{px}} \times L}$
Let $D = [\lambda_1, \ldots, \lambda_K] \in \mathbb{R}^K$ and $V = [v_1, \ldots, v_K] \in \mathbb{C}^{M \times K}$
$E \leftarrow \Phi W V$ // $E \in \mathbb{C}^{N_{\text{px}} \times K}$
$I_{\widetilde{\mathcal{S}}} = (E \circ \overline{E}) D$ // $(E \circ \overline{E})_{ij} = E_{ij}\overline{E}_{ij}$

As mentioned in Section 2.2, one of Bluebild's main advantages is its ability to provide an energy-level view of the sky. This is portrayed in Figure 6 where 3 sources are closely packed in a narrow field-of-view, two of which cannot be easily distinguished due to their proximity. Moreover, with a signal-to-noise ratio of −10dB, non-source positions in the true estimate 6a severely pollute the intensity map. However, the 3 leading eigenfunctions 6b, 6c, 6d are void of artefacts and each contain one of the sources located in the field-of-view. As there are only 3 sources in the sky, the trailing eigenfunctions



(a) Full-sky image of the northern hemisphere. A single source with $\sigma^2 = 1$ is located at $d = e_3$.



(b) 45° FoV image centered at $d = e_1 + e_2$. Three sources with $\sigma^2 = [0.65, 0.75, 1]$ are randomly scattered around d.

Fig. 5. Sound-field image $I_{\widetilde{S}}$ using Bluebild with $W = I_L$. 50ms of audio samples at $f_c = 1.5$kHz are obtained using a 48-element spherical array with $f_s = 48$kHz. The signal-to-noise ratio (SNR) at the microphones is 10dB. True source positions are enclosed in black circles.

6e all correspond to noise, thus we can simply discard them to obtain a denoised estimate 6f for $I_{\widetilde{S}}$.

2.4 Resolution & Beamforming

In Figures 5 and 6, it is interesting to notice that although we assumed a point-source model, the intensity estimate $I_{\widetilde{S}}$ is not composed of point-sources. Instead, each source is characterized by a compact blob, hence resolving close sources as in Figure 6a can become problematic. This blurring effect can be linked to the *point-spread function* [16] (PSF) of the acquisition device, and can be shown to act on the estimated process $\widetilde{\mathcal{S}}$ and $I_{\widetilde{S}}(r)$:

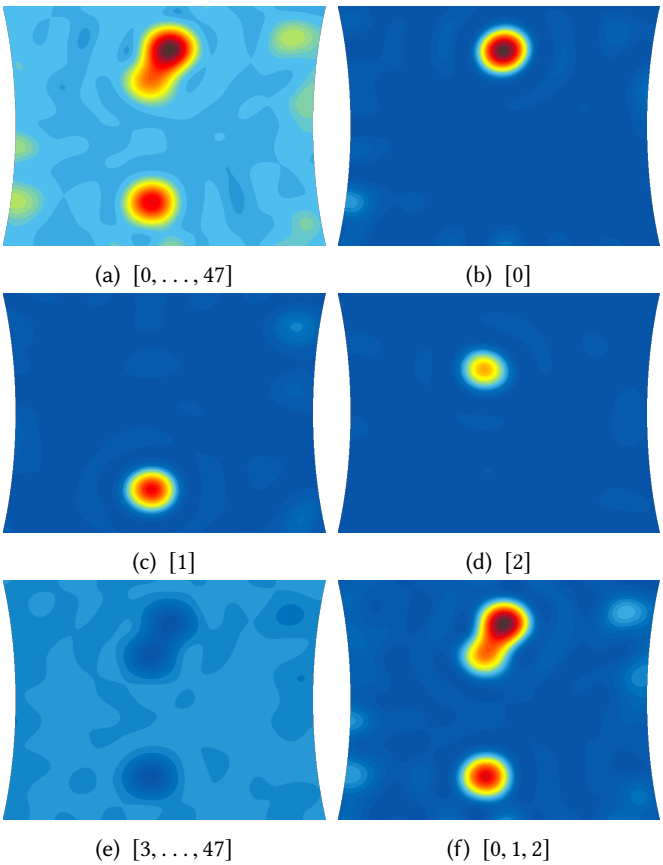
$$\begin{aligned}
\widetilde{S}(r) &= \sum_{m=1}^{M} Y_m \widetilde{\psi}_m(r) = \sum_{m=1}^{M} \langle \mathcal{S}, \psi_m \rangle_{\mathbb{S}^2} \widetilde{\psi}_m(r) \\
&= \sum_{m=1}^{M} \int_{\mathbb{S}^2} S(u) \psi_m^*(u) du \; \widetilde{\psi}_m(r) = \int_{\mathbb{S}^2} S(u) \sum_{m=1}^{M} \widetilde{\psi}_m(r) \psi_m^*(u) du \\
&= \int_{\mathbb{S}^2} S(u) \eta(r, u) du = \langle \mathcal{S}, \eta(r, \cdot) \rangle_{\mathbb{S}^2}, \qquad (14)
\end{aligned}$$

$$\begin{aligned}
I_{\widetilde{S}}(r) &= \mathbb{E}\left[\widetilde{S}(r)\widetilde{S}^*(r)\right] \\
&= \mathbb{E}\left[\int_{\mathbb{S}^2} S(u)\eta(r,u)du \int_{\mathbb{S}^2} S^*(v)\eta^*(r,v)dv\right] \\
&= \int_{\mathbb{S}^2 \times \mathbb{S}^2} \kappa_{\mathcal{S}}(u,v)\eta(r,u)\eta^*(r,v)dudv, \qquad (15)
\end{aligned}$$

where

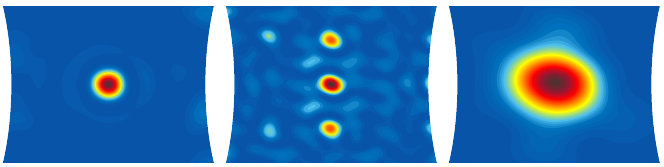
$$\begin{aligned}
\eta(\cdot, u) : \mathbb{S}^2 &\to \mathbb{C} \\
r &\to \sum_{m=1}^{M} \widetilde{\psi}_m^*(r) \psi_m(u) \qquad (16)
\end{aligned}$$

is the point-spread function of the instrument at $u \in \mathbb{S}^2$. Using (12) and $\kappa_{\mathcal{S}}(r_1, r_2) = \sum_{q=1}^{Q} \sigma_q^2 \delta(r_1 - r_q)\delta(r_2 - r_q)$, (15) simplifies



(a) $[0, \ldots, 47]$ (b) $[0]$ (c) $[1]$ (d) $[2]$ (e) $[3, \ldots, 47]$ (f) $[0, 1, 2]$

Fig. 6. Sound-field images $I_{\widetilde{S}}$ using Bluebild with $W = I_L$ and different spectrum truncations. 50ms of audio samples at $f_c = 1.5$kHz are obtained using a 48-element spherical array with $f_s = 48$kHz. The signal-to-noise ratio (SNR) at the microphones is -10dB. Three sources with $\sigma^2 = [0.75, 0.5, 1]$ are located in a 45° FoV centered around $d = e_1 + e_2$, with the last two being only 6.37° apart. In each case, $I_{\widetilde{S}}$ only contains the eigenfunctions indicated in square brackets. Notice that sources having different energy levels are located in distinct eigenfunctions. Moreover, truncating the spectrum to the first 3 eigenfunctions denoises the estimate (Figure 6f).



(a) 48-element spherical array of radius 1. (b) 48-element cube array. (c) 48-element gaussian array with $\sigma^2 = 0.125$.

Fig. 7. Effect of antenna geometry on the point-spread function for spherical, cubic and Gaussian arrays. Regular geometries like 7b cause grating lobes that are the source of artefacts in $I_{\widetilde{\mathcal{S}}}$. Moreover, antennas must be sufficiently spaced apart to image a given frequency range.

as follows:

$$\begin{aligned} I_{\widetilde{\mathcal{S}}}(r) &= \int_{\mathbb{S}^2\times\mathbb{S}^2} \kappa_{\mathcal{S}}(u,v)\eta(r,u)\eta^*(r,v)dudv \\ &= \sum_{q=1}^{Q} \sigma_q^2 \int_{\mathbb{S}^2} \eta(r,u)\delta(u-r_q)du \int_{\mathbb{S}^2} \eta^*(r,v)\delta(v-r_q)dv \\ &= \sum_{q=1}^{Q} \sigma_q^2 \eta(r,r_q)\eta^*(r,r_q) = \sum_{q=1}^{Q} \sigma_q^2 |\eta(r,r_q)|^2. \end{aligned}$$

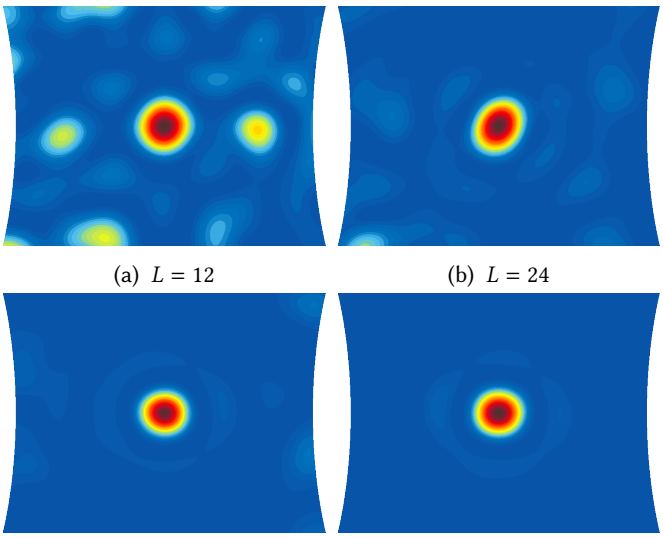
In other words, the point-spread function acts as a smoother on $I_{\mathcal{S}}$, smearing out the point sources into blobs, limiting the resolution of the instrument. A sample PSF is shown in Figure 7a for a 48-element spherical array.

Although it is not possible to escape from the effect of the PSF on $I_{\widetilde{\mathcal{S}}}$, several parameters can be tuned to obtain desirable shapes. First, the geometry of the antenna array $\{p_l,\ l \in \{1,\dots,L\}\}$ plays an important role as can be seen in Figure 7 where the PSFs of spherical, cubic, and Gaussian arrays are portrayed. In addition to having much higher sidelobes than the spherical array, the cubic array presents two strong grating lobes that cause severe artefacts in intensity estimates. Aside from shape, the instrument must be correctly scaled w.r.t. the wavelength of interest. For example, Figure 7c shows what happens when the antennas are too close given that the wavefronts are centered at $f_c = 1.5$kHz.

Assuming the instrument is correctly sized (in terms of antenna spacing), we can try to smoothen the PSF by increasing the total number of antennas L. Figure 8 shows the evolution of the PSF as the number of antennas in a spherical array increases: the main-lobe becomes more compact and side-lobes are reduced. Note however that arbitrarily increasing L can make the Gram ill-conditioned such that additional artefacts appear in computed PSFs because of numerical instabilities.

Finally, the resolution of the instrument can be modified by judiciously choosing the beamforming matrix $W \in \mathbb{C}^{L\times M}$. Indeed, we have

$$\begin{aligned} Y_m = \langle X, W_m\rangle &= \sum_{l=1}^{L} X_l W_{lm}^* \\ &= \langle \mathcal{S}, \sum_{l=1}^{L} W_{lm}\phi_l\rangle_{\mathbb{S}^2} \\ &= \langle \mathcal{S}, \psi_m\rangle_{\mathbb{S}^2}, \end{aligned}$$



(a) $L = 12$ (b) $L = 24$ (c) $L = 48$ (d) $L = 96$

Fig. 8. Effect of number of antennas on the point-spread function of an L-element spherical array. (We assume $W = I_L$.) As L increases, the main lobe shrinks and sidelobes are reduced.

where W_m denotes the m-th column of W. Beamforming can therefore be seen as transforming the L physical antennas into $M \leq L$ virtual antennas having different filtering kernels $\{\psi_m, m \in \{1,\dots,M\}\}$ [15]. For example, one can choose matched-beamforming weights to achieve high spatial resolution in a small field-of-view [1, 15], at the cost of strong sidelobe structures. Alternatively, we can trade high localized resolution for sensitivity over a wider field-of-view as can be seen in Figure 9, or use randomized beamforming techniques [1] to maximize the information content coming from all directions.

In the end, optimal instrument design requires one to tune the geometry, size and beamforming of the array to the application of interest.

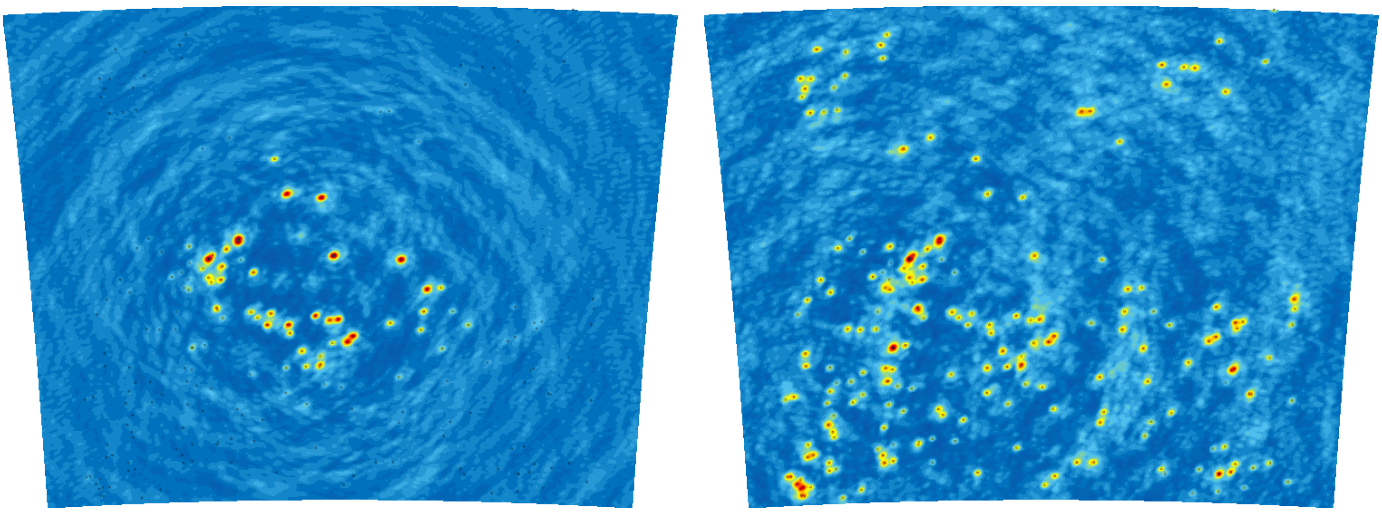
2.5 Imaging by Beamforming

Given that beamforming acts as a spatial filter on the random field $\mathcal{S}$, one can estimate the intensity map $I_{\mathcal{S}}$ by spatially scanninng the sky using beamforming. Concretely, assuming a Q point-source model as in (12) and instrument model 2.2, antenna samples $\{X_l,\ l \in \{1,\dots,L\}\}$ take the form

$$\begin{aligned} X_l &= \langle \mathcal{S}, \phi_l\rangle_{\mathbb{S}^2} \\ &= \int_{\mathbb{S}^2} \sum_{q=1}^{Q} \xi_q \delta(r-r_q) e^{-j\frac{2\pi}{\lambda_c}\langle p_l, r\rangle} dr \\ &= \sum_{q=1}^{Q} \xi_q e^{-j\frac{2\pi}{\lambda_c}\langle p_l, r_q\rangle} \\ &= \Phi_l^H F, \end{aligned}$$

where Φ_l is the l-th column of $\Phi \in \mathbb{C}^{Q\times L}$ such that $\Phi_{ql} = \phi(r_q, p_l) = e^{j\frac{2\pi}{\lambda_c}\langle p_l, r_q\rangle}$ and $F = \left[\xi_1,\dots,\xi_Q\right] \in \mathbb{C}^Q$. The beamformed antenna samples $Y \in \mathbb{C}^M$ can therefore be written as

$$Y = W^H\left(\Phi^H F + \widetilde{N}\right) = \Psi^H F + N, \tag{17}$$



(a) Matched beamforming

(b) Gaussian beam constructed with Flexibeam.

Fig. 9. Using Bluebild to estimate the intensity of a 12° patch of the radio-sky containing 200 sources. Narrow beamshapes like matched beamforming (MB) achieve high resolution but have low spatial sensitivity. Using the Flexibeam framework [10], wider beams can be constructed that trade resolution for spatial sensitivity as seen above.

where $N \sim \mathcal{N}_c\left(0, \sigma^2 W^H W\right)$ is the correlated noise introduced at the virtual antennas. Imaging by beamforming [1, 18] consists in beamforming Y with a user-chosen beam-steering vector $b(r) \in \mathbb{C}^M$, then computing the variance of the random field at $r \in \mathbb{S}^2$ as

$$I_{\mathcal{S}}(r) = \mathbb{E}\left[b^H(r) Y Y^H b(r)\right] = b^H(r) \Sigma b(r). \tag{18}$$

As such, different choices of $b : \mathbb{S}^2 \to \mathbb{C}^M$ will lead to different intensity estimates for $I_{\mathcal{S}}$. Below we describe three common choices for $b(r)$ and study their properties.

B-scan. Beamformed-scan [12] chooses $b(r) = \Psi^*(r)$, where $\Psi(r) = [\psi_1(r), \dots, \psi_M(r)]^T$. With this choice of beam-weights, $I_{\mathcal{S}}$ takes the form

$$I_{\mathcal{S}}(r) = \Psi^H(r) \Sigma \Psi(r). \tag{19}$$

Notice the similarity of (19) with the least-squares estimate (9):

$$I_{\widetilde{\mathcal{S}}}(r) = \Psi^H(r) G_\Psi^{-1} \Sigma G_\Psi^{-1} \Psi(r).$$

As such, B-scan can be seen as choosing the interpolation operator as Ψ, adjoint of the sampling operator Ψ^*. This is problematic because from (13) we know that $\Psi^* \Psi = G_\Psi \neq I_M$, so although $P = \Psi \Psi^*$ is self-adjoint, it is not idempotent and hence not a projection operator, leading to sub-optimal estimates in the least-squares sense. Due to this, the Bluebild least-squares estimate $I_{\widetilde{\mathcal{S}}}$ can be very far from the B-scan estimate $I_{\mathcal{S}}$ if G_Ψ is ill-conditioned. This can be seen in Figure 10 where the PSFs of Bluebild and B-scan already differ significantly for cond $(G_\Psi) \approx 34$. The problem only worsens as the conditioninng of G_Ψ increases.

MVDR. Minimum Variance Distortionless Response [12, 18] chooses beam-weights to satisfy the following optimization criteria:

$$\begin{aligned} b(r) = \arg\min_b \quad & b^H \Sigma b \\ \text{subject to} \quad & b^H \Psi(r) = 1. \end{aligned}$$

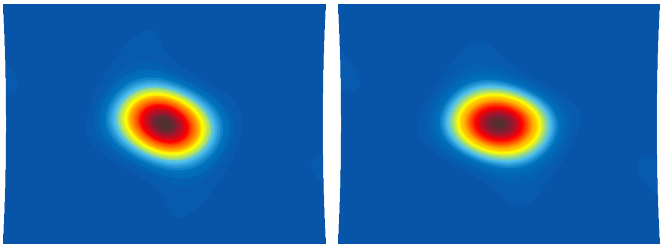
In other words, $b(r)$ is chosen to minimize the energy contribution from directions other than r, while maintaining a fixed response in direction r. The problem has the closed form solution

$$b(r) = \frac{\Sigma^{-1} \Psi(r)}{\Psi^H(r) \Sigma^{-1} \Psi(r)},$$

such that the intensity estimate can be rewritten as

$$I_{\mathcal{S}}(r) = \frac{1}{\Psi^H(r) \Sigma^{-1} \Psi(r)}.$$

Despite MVDR's good performance at high SNR as seen in Figure 11, its estimates quickly worsen as the noise floor increases. On the contrary, Bluebild estimates lie almost constant at all levels of SNR. Moreover, while Bluebild's intensity estimates seem to have a higher noise floor, we can leverage the functional PCA decomposition to denoise the estimate. In addition, due to its reliance on Σ, MVDR is a data-dependent beamformer which may not be desirable. All in all, algorithms such as Bluebild seem particularly interesting in applications such as radio-astronomy where low SNR dominates.



(a) Bluebild

(b) B-scan

Fig. 10. Point-spread functions of Bluebild (left) and B-scan (right) on a 22° field-of-view. The microphone array consists of 96 Gaussian-distributed microphones with $W = I_L$ such that cond$(G_\Psi) \approx 34$. Even with mild conditioning, both PSFs differ significantly.

MUSIC. MUltiple SIgnal Classification [14] is a subspace-based method to localize sources. It consists of two stages:

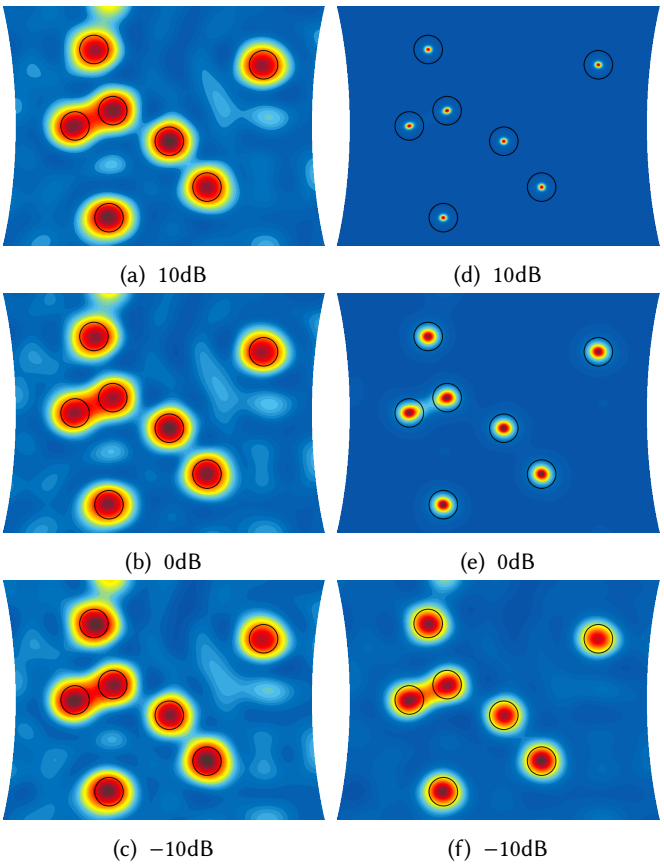


Fig. 11. Sound-field image $I(r)$ using Bluebild (left) and MVDR (right) with $W = I_L$ at different signal-to-noise ratios. 50ms of audio samples at $f_c = 1.5$kHz are obtained using a 48-element spherical array with $f_s = 48$kHz. True source positions are enclosed in black circles. In high SNR, MVDR fairs much better than Bluebild given the instrument geometry, but its estimates quickly worsen as SNR decreases. Bluebild on the other hand is stable throughout the entire SNR range, even without spectral truncation.

(1) Decompose the empirical correlation matrix $\widehat{\Sigma} = YY^H$ into signal and noise components by finding the solutions (λ_m, v_m) of the generalized eigenvalue problem

$$\widehat{\Sigma} v = \lambda \Sigma_N v,$$

where $\Sigma_N = \sigma^2 W^H W$ is the correlation matrix of the noise process $N \in \mathbb{C}^M$. The eigenvalues $\{\lambda_m,\ m \in \{1, \dots, M\}\}$ are clustered into two groups where the cluster associated with the smallest eigenvalues corresponds to noise components in the signal. Let D_S be the diagonal matrix containing the eigenvalues corresponding to the signal subspace, and E_S their associated eigenvectors. Let D_N, E_N be the equivalent quantities corresponding to the noise subspace.

(2) Form the pseudo-spectrum $P_{\text{MU}} : \mathbb{S}^2 \to \mathbb{R}$, peaks of which correspond to true source locations:

$$P_{\text{MU}}(r) = \frac{1}{\left\| E_N^H \Psi(r) \right\|^2}.$$

The idea behind the pseudo-spectrum is that signals coming from directions $\{r_q,\ q \in \{1, \dots, Q\}\}$ are not part of the noise subspace, hence the projection of the response vector $\Psi(r_q)$ onto the noise subspace E_N should have small norm, i.e. $P_{\text{MU}}(r_q)$ will be very large.

The pseudo-spectrum $P_{\text{MU}}(r)$ is shown for different SNR levels in Figure 12g-12i. It is easy to see that all sources are precisely located and resolved.

It is important to note however that $P_{\text{MU}}(r)$ is *not* $I_{\mathcal{S}}(r)$, but a non-linear view of the resemblance of signals from direction $r \in \mathbb{S}^2$ to noise. Comparing MUSIC to the likes of B-scan, Bluebild and MVDR with the current formulation is therefore unfair. From the perspective of (18), what MUSIC tries to achieve is spatially filtering samples $Y \in \mathbb{C}^M$ to only keep the contributions that lie in the signal subspace E_S, followed by an interpolation step using $\Psi(r)$. In other terms, we want

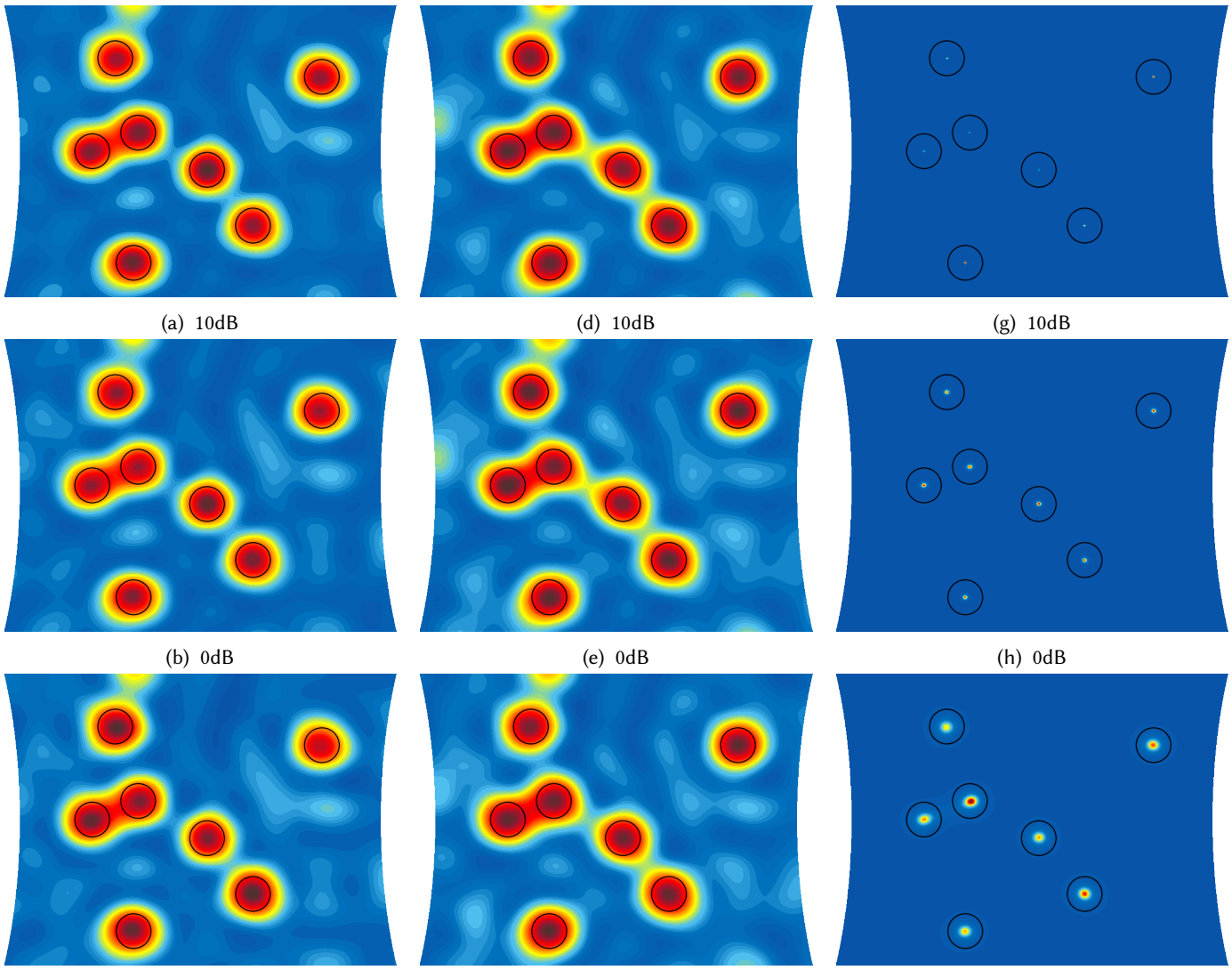
$$I_{\mathcal{S}}(r) = \Psi^H(r) E_S E_S^H \Sigma E_S^H E_S \Psi(r) = b^H(r) \Sigma b(r),$$

which corresponds to B-scan preceded with a filtering stage, also known as principle component beamforming [4, 12]. The beam-weights achieving this is $b(r) = E_S^H E_S \Psi(r)$, and reconstructions $I_{\mathcal{S}}(r)$ using MUSIC are shown in Figure 12d-12f, where we manually force MUSIC to correctly identify the number of sources Q. Due to this, Bluebild estimates in 12a-12c have had their spectrums truncated to the seven leading eigenvalues to remain fair. Like Bluebild, both MUSIC's estimate of $I_{\mathcal{S}}$ and the pseudo-spectrum exhibit strong resilience to noise, but similar to MVDR, the beam-weights are data-dependent which may be undesirable. Although MUSIC's performance is very good, being a subspace method means that MUSIC can at most find $Q \leq M - 1$ sources in $I_{\mathcal{S}}$ or P_{MU}. Radio-astronomy applications like Figure 9 are therefore out of its reach as even narrow fields-of-view can contain $Q \gg M$ sources.

3 ALGORITHM INSIGHTS FOR REAL-TIME IMAGING

Now that Bluebild's theoretical properties are known, we seek to use the algorithm in the context of sound field imaging and radio-interferometry to obtain estimates of the intensity function $I_{\mathcal{S}}$, with the twist that the algorithm run in real-time. This is important for multiple reasons:

- In the acoustic context, the random field $\mathcal{S}$ is generally not stationary on long time intervals: acoustic sources, namely travelers as in Figure 3, move through time and emit meaningful voice signals that (hopefully) make grammatical sense. As such, any localization algorithm based on source stationarity needs to run multiple times per second to show the evolution of $I_{\widetilde{\mathcal{S}}}$.
- In the radio-astronomy context, the problem is reversed: the radio-sky $\mathcal{S}$ is usually stationary, but due to the rotation of the Earth, the sampling operator Ψ^* changes through time. Hence, virtual-antenna samples $Y(t) \in \mathbb{C}^M$ are not stationary. The intensity map $I_{\widetilde{\mathcal{S}}}$ is therefore computed on short time-intervals during which $\Sigma \in \mathbb{C}^{M \times M}$ is assumed stationary, typically between 8 and 30 seconds, then summed together.
- On another note, the typical workflow in radio-astronomy is to acquire all the data from observations, then do the imaging offline. Although manageable for current-generation telescopes like LOFAR [6], next-generation instruments like the SKA will generate data in excess of 100Tb/s [7], all of which can't be stored realistically. As



(a) 10dB (d) 10dB (g) 10dB

(b) 0dB (e) 0dB (h) 0dB

(c) −10dB (f) −10dB (i) −10dB

Fig. 12. Sound-field image $I(r)$ using Bluebild (left), MUSIC (middle), and the MUSIC pseudo-spectrum P_{MU} (right) with $W = I_L$ at different signal-to-noise ratios. 50ms of audio samples at $f_c = 1.5$kHz are obtained using a 48-element spherical array with $f_s = 48$kHz. True source positions are enclosed in black circles. We manually tell MUSIC the dimensionality of the signal space, hence Bluebild estimates are also truncated to the first Q eigenfunctions.

such, real-time stream-computing of $I_{\widetilde{S}}$ could drastically cut down the data to store.

In light of these constraints, the goal of this chapter is to develop algorithmic ideas that leverage domain-specific knowledge from acoustics and radio-astronomy to achieve real-time interferometric imaging with Bluebild. Before diving into both applications however, we start off by taking a closer look at the problem from a complexity point-of-view.

3.1 A Tale of Two Complexities

Prior to optimizing an algorithm's execution speed, it is important to do a theoretical complexity analysis to identify and tackle potential bottlenecks. However, one must distinguish the complexity of an algorithm from its run-time performance. In the case of Bluebild as described in Algorithm 2, the breakdown of each step is given in Table 1, where for simplicity we assume $K = M$ and take $M \ll L \ll N_{\text{px}}$ to form the final aggregate complexity.

From the analysis, it seems that the most expensive operation would be computing $E = \Phi(WV)$, but Table 2 paints a different picture. Despite the higher complexity, E is very cheap to compute in practice compared to Φ. This is because E only depends on floating-point arithmetic whereas Φ requires the computation of complex exponentials that are inherently slower to estimate.

The lesson here is that both complexity and run-time performance should be taken into account during optimization since not all operations are weighted equally.

3.2 Acoustics

In the acoustic context, our goal is to solve the robot's dilemma in Figure 3: make real-time estimates of the sound field intensity $I_{\widetilde{S}}$ using Bluebild to locate travelers in the train station. As mentioned earlier, the Bluebild estimate must be computed on small time intervals during which the voice signals can be assumed stationary. However, source model 2.1 also assumes

Table 1. Theoretical complexity breakdown of Algorithm 2 where we assume $K = M$ for simplicity. For the total aggregate complexity, we assume $M \ll L$ as is common in radio-astronomy.

Operation	Description	Complexity
$\widehat{\Sigma}$	$\frac{1}{N_s} YY^H$	M^2
G_Ψ	$W^H \Phi^* \Phi W$	$L^2 + ML^2 + M^2L$
(D, V)	$\widehat{\Sigma} v_m = \lambda_m G_\Psi v_m$	M^3
Φ	$e^{j\frac{2\pi}{\lambda_c} x_{\text{grid}} P^T}$	$N_{\text{px}}L + N_{\text{px}}L$
E	$\Phi(WV)$	$LM^3 + N_{\text{px}}LM$
$I_{\widetilde{S}}$	$(E \circ \overline{E})D$	$N_{\text{px}}M + N_{\text{px}}M$
Total		$O\left(L^2M^3 + N_{\text{px}}LM\right)$

Table 2. Run-time breakdown of Algorithm 2 on an Intel i7-7600U where we choose $K = M$ for simplicity. The pair (M, L) is set to $(62, 1488)$, matching the HBA antenna configuration of the LOFAR telescope. run-time estimates for each step are normalized w.r.t. total run-time and correspond to the average of 50 independent runs. As N_{px} increases, the complexity fully transfers onto Φ and E. Contrary to the theoretical complexity where E is shown to cost more that Φ, the reverse is true in practice. The same trends hold for other Intel processor families and the IBM POWER8 architecture.

	N_{px}						
Operation	64^2	96^2	128^2	192^2	256^2	384^2	512^2
$\widehat{\Sigma}$	0.3	0.1	0.1	0	0	0	0
G_Ψ	24.6	12.8	7.7	3.6	2	0.9	0.5
(D, V)	0.8	0.4	0.2	0.1	0.1	0	0
Φ	66.6	77.5	82.9	86.5	87.7	88.6	88.6
E	7.6	8.5	8.7	9.4	10	10.2	10.5
$I_{\widetilde{S}}$	0.2	0.3	0.3	0.3	0.3	0.3	0.3

incident signals are narrow-band. Therefore, one needs to also split the voice frequency range into narrow subbands and form a Bluebird estimate per subband. To summarize, Algorithm 3 shows the necessary steps to get a multi-frequency Bluebird estimate of the sound field, and Figure 13 shows sample snapshot images obtained using Algorithm 3 at different frequency bands.

Unfortunately, Algorithm 3 is too slow to do real-time imaging at our intended rates: For the source tracking to be fluid, we need to image the sound field approximately 10 times per second, i.e. have an image rate of 10 frames-per-second (FPS). At the same time, knowing $I_{\widetilde{S}}$ at different frequency bands could provide additional information to distinguish between closely-packed sources, thus we would like the number of subbands B to be relatively large. However, simulations using the same setup as Figure 13 show that we can only image the sound field at 2 FPS with $B = 9$. The Bluebird estimate therefore needs to be computed substantially faster to be usable in practice, but how can we achieve this?

As seen in Table 2, the bulk execution time lies in computing Φ, but notice on line 6 of Algorithm 3 that only a subset of Bluebird's parameters change through time, namely frequency samples $Y_{\mathcal{F}}^{B,t}$ and eventually the beam-weights W_b. This leads to a number of observations:

Algorithm 3: Multi-Frequency-Bluebird(Y, P, W, τ, λ, x_{grid})

Input: $Y \in \mathbb{C}^{M \times \mathbb{N}}$: instrument samples
Input: $P \in \mathbb{R}^{L \times 3}$: antenna positions
Input: $W = \left\{W_b \in \mathbb{C}^{L \times M},\ b \in \{1, \ldots, B\}\right\}$: beamforming weights per subband
Input: $\tau = \{\tau_b \in [0, 1],\ b \in \{1, \ldots, B\}\}$: energy threshold per subband
Input: $\lambda = \{\lambda_b \in \mathbb{R},\ b \in \{1, \ldots, B\}\}$: source signal wavelength per subband
Input: $x_{\text{grid}} \in \mathbb{R}^{N_{\text{px}} \times 3}$: grid-points in Cartesian coordinates
Output: $I_{\widetilde{S}} = \left\{I_{bt} \in \mathbb{R}^{N_{\text{px}}},\ (b, t) \in \{1, \ldots, B\} \times \mathbb{N}\right\}$: intensity function per subband and short time interval

1 Bin time-samples Y into short time intervals of length N:

$$\mathcal{Y} = \left\{Y^t \in \mathbb{C}^{M \times N},\ t \in \mathbb{N}\right\}$$

2 **for** $t \in \mathbb{N}$ **do**
3 Compute spectrum $Y_{\mathcal{F}}^t$ of Y^t:

$$Y_{\mathcal{F}}^t \leftarrow \text{DFT}(Y^t) \in \mathbb{C}^{M \times N},$$

$$Y_{\mathcal{F}}^t[:, k] = \sum_{n=1}^{N} Y^t[:, n] e^{-j\frac{2\pi}{N}kn}, \quad k \in \{1, \ldots, N\}$$

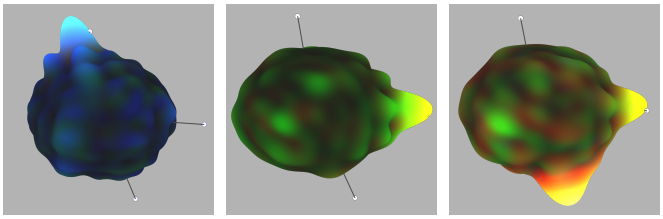
4 Aggregate frequencies into B disjoint subbands $\{B_1, \ldots, B_B\}$ to obtain $Y_{\mathcal{F}}^{B,t} \in \mathbb{C}^{M \times B}$:

$$Y_{\mathcal{F}}^{B,t}[:, b] = \sum_{k \in B_b} Y_{\mathcal{F}}^t[:, k], \quad b \in \{1, \ldots, B\}$$

5 **for** $b = \{1, \ldots, B\}$ **do**
6 Form Bluebird estimate using Algorithm 2:

$$I_{bt} \leftarrow \text{Bluebird}\left(Y_{\mathcal{F}}^{B,t}[:, b],\ P,\ W_b,\ \tau_b,\ \lambda_b,\ x_{\text{grid}}\right)$$

7 **end**
8 **end**



(a) Speaker 1 (b) Speaker 2 (c) Speakers 2 and 3

Fig. 13. Snapshot images of the sound field using Algorithm 3 and $N_{\text{px}} = 128^2$. Three speakers having different voice-pitches are located in a room and take turns to talk either alone (13a and 13b) or simultaneously (13c). Audio samples are recorded with the Pyramic 48-element microphone array. The 1500-4500Hz frequency range is split into $B = 9$ subbands, each of which is imaged separately with Bluebird. To obtain RGB images above where red, green, and blue colors represent low-, mid-, and high-frequency signals respectively, the 9 Bluebird estimates are summed together in groups of 3 subbands. The 3 white markers correspond to the true source locations. Data from [13].

Table 3. Effect of spectral truncation on execution time when using Algorithm 3 with and without the real-time (RT) optimizations described in Section 3.2. We assume the same setup as in Figure 13, i.e. $(M, L, B) = (48, 48, 9)$. The numbers show how much time is needed to compute 10 time-snapshots per second, hence any number above 1 is *not* real-time. We see that spectral truncation leads to massive speedups when using the RT optimizations, but has a negligeable effect when using Algorithm 3 without them.

K	Algorithm 3 (sec)	Algorithm 3 RT (sec)
1	4.73	0.13
8	4.86	0.24
16	4.89	0.34
32	5.08	0.56
48	5.13	0.93

- The antenna geometry P and grid-points x_{grid} are time-invariant, hence we can compute ahead of time the set $\Phi = \{\Phi_b,\ b \in \{1, \dots, B\}\}$. By shifting Φ to a pre-computation, the run-time is relieved of 80% of its compute bottleneck.
- With Φ gone from run-time, Table 2 shows that the execution time shifts to G_Ψ, E and $I_{\widetilde{\mathcal{S}}}$.
- In G_Ψ, the $\Phi^*\Phi$ term only depends on microphone positions P and the subband wavelength λ_b, hence G_Ψ can be partially computed offline: the only part that will take place at run-time is left- and right- multiplication by W_b.
- E and $I_{\widetilde{\mathcal{S}}}$ depend on K, which itself depends on τ. Even in low SNR, the spectrum can be safely truncated without much loss in reconstruction quality. As the execution time of E and $I_{\widetilde{\mathcal{S}}}$ depends linearly on K, any truncation will have significant benefits on execution time. As Table 3 shows, this is certainly *not* true if Φ were also computed online due to the relative importance of E and $I_{\widetilde{\mathcal{S}}}$ on total execution time.

With these optimizations in hand, simulation results from Table 3 show that 10 FPS is easily achievable, with the potential to reach much higher rates by spectrum truncation.

3.2.1 Building a Real-Time Sound-Field Imager. Given the conclusive simulation results obtained, we set out to build a demo application with a microphone array to see if the simulation results would translate well to the real-world. A brief description of the hardware setup is given below:

- The setup consists of a laptop and the Pyramic [2] array, connected together through a TCP connection.
- Microphone samples are acquired and streamed in real-time to the laptop using the easy-dsp [5] package.
- On the laptop, a Python implementation of Algorithm 3 with RT extensions processes data packets $Y^t \in \mathbb{R}^{M \times N}$ and forms Bluebild images on the fly.
- Due to visualization issues encountered with Python, computed images are then sent over a TCP connection to a MATLAB process that shows the images of the sound field on the screen.

A flowchart with the main steps of the implementation is shown in Figure 14a.

Figures 14b-14d show the experimental setup of the demo along with illustrations of the reconstructed sound fields: The demo takes place in an auditorium, where the Pyramic array is located roughly 2 meters away from the closest wall (white background in Figures 14b-14d). A user actively circles around the array holding a phone playing a highly-dynamic piece of music containing voice signals.

Figure 14b shows a similar RGB spectrum to those present in Figure 13, where the direction of maximum amplitude corresponds to the source location, and colors denote the spectral content of the signals in the 1500-4500Hz range. As the direction of maximum amplitude is colored white, it means that the source emits frequencies that span the full frequency range. Notice the ripples in the image which are caused by the side-lobe structures of the PSF, considerably polluting the intensity estimate.

Figure 14c shows the same information as 14b, but presented differently: Instead of visualizing $I_{\widetilde{\mathcal{S}}}$ per frequency band, we are instead interested in tracking the source as it moves around the array. To do so, all B intensity estimates are summed together to obtain a multi-frequency estimation of $I_{\widetilde{\mathcal{S}}}$. Moreover, to avoid having PSF ripples in the estimate, we take advantage of Bluebild's post-processing capabilities discussed in Section 2.2 to soft-threshold the eigenfunctions of $I_{\widetilde{\mathcal{S}}}$ before summing them together. The result is an intensity estimate that only shows the dominant peaks of $I_{\widetilde{\mathcal{S}}}$, making it very easy to track sources as they move around the sphere.

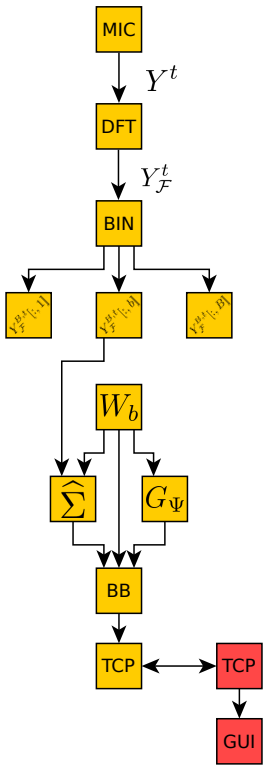
Finally, Figure 14d illustrates our ability to track sources located in different subbands.

In terms of speed, our implementation is capable of imaging the sound field at 10 FPS when $B = 9$, with slight fluctuations depending on machine load, hence the simulation results do translate well to real-world situations. However, although we can compute enough frames to have a fluid view of the sound field, the MATLAB process can only show up to 5 FPS due to rendering limitations. In order to avoid having packet accumulation and thus have the visualisation lag behind the user's movements, we artificially limit the frame-rate to 5 FPS in the Python implementation. The demo was successfully presented at the demo session of ICASSP 2017 [3].

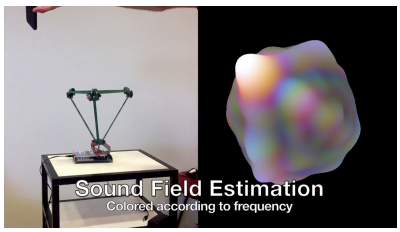
3.3 Radio-astronomy

In the radio-astronomy context, our goal is to form estimates of the sky intensity function $I_{\widetilde{\mathcal{S}}}$ in real-time. However, significant differences to the acoustic context of Section 3.2 mean that most optimizations applied in acoustics cannot be transferred to radio-astronomy. The main differences between both applications are detailed below:

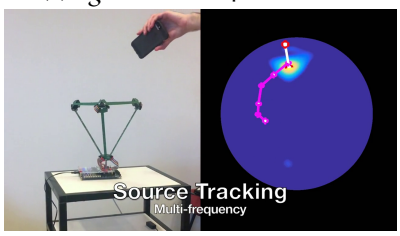
Instrument mechanics. Compared to acoustics, the source density in radio-astronomy is very high, with small fields-of-view containing potentially thousands of sources. As such, radio-telescopes must use a large number of antennas distributed over a wide area to achieve high angular resolution. From an engineering perspective, simply building a system to handle the huge data-streams from the antennas is daunting, thus limitations are imposed on the operating modes of the instrument to make it practically feasible. Concretely, modern radio-telescopes adopt a hierarchical design: geographically close antennas are grouped into stations where antenna signals are beamformed together to obtain a unique data-stream per station (Figure 15). As such, up to a permutation of the



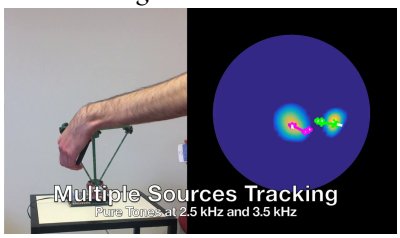
(a) System block diagram.



(b) $I_{\widetilde{\mathcal{S}}}$ color-coded per subband.



(c) Using $I_{\widetilde{\mathcal{S}}}$ to do source tracking.



(d) Tracking two sources located in different subbands.

Fig. 14. 14a: block diagram detailing the main steps of the real-time implementation of Bluebild. Each audio packet is split into frequency components and distributed over B frequency bins. Each bin is then imaged using Bluebild with RT extensions. Upon completion, images are sent to MATLAB through a TCP socket for visualization (red blocks in the figure). Block labels mimic those of Algorithm 3. 14b-14d: side-by-side images of the ground truth and Bluebild estimates $I_{\widetilde{\mathcal{S}}}$. A video of the demo is available in [11].

antennas, the beamforming matrix $W \in \mathbb{C}^{L\times M}$ can be written in block-diagonal form (Figure 16) and is therefore extremely sparse.

Snapshot Imaging vs. Long-Exposure Imaging. Due to the low SNR of sky signals, taking a single snapshot image during a stationary interval yields very poor images that suffer both from noise artefacts and the coherent superposition of sidelobe artefacts from all sources, such as those induced by grating lobes (Figure 7b). It is therefore common to do long-exposure imaging: at each integration time, choose beamweights $W \in \mathbb{C}^{L\times M}$ to continuously focus the beam on the same direction $f \in \mathbb{S}^2$. Once all snapshots $I_{\widetilde{\mathcal{S}}} = \left\{I^t_{\widetilde{\mathcal{S}}},\ t \in \mathcal{T}\right\}$ have been taken, the final long-exposure image is given by

$$I^{\mathcal{T}}_{\widetilde{\mathcal{S}}} = \frac{1}{\mathrm{card}(\mathcal{T})} \sum_{t\in\mathcal{T}} I^t_{\widetilde{\mathcal{S}}},$$

where noise artefacts are significantly reduced. One would assume however that long-exposure imaging will *not* remove the sidelobe artefacts as they would still coherently superpose and pollute the intensity estimate. But as Figure 17 shows, the PSF of the instrument rotates through time, hence snapshots both have uncorrelated noise samples and non-overlapping

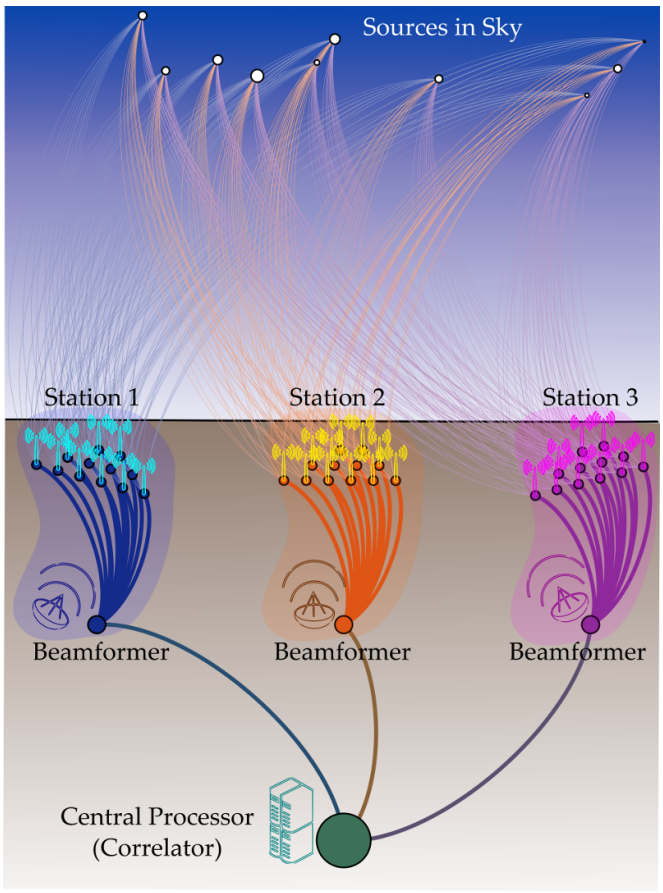


Fig. 15. Schematic representation of a hierarchical design for modern radio telescopes. Signals received by individual antennas are beamformed at the station level before being sent to the central processor for further processing. (Image from [15].)

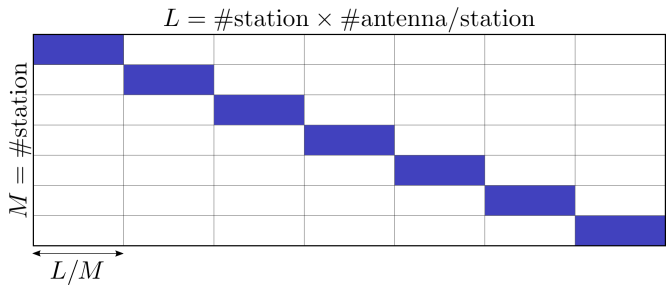
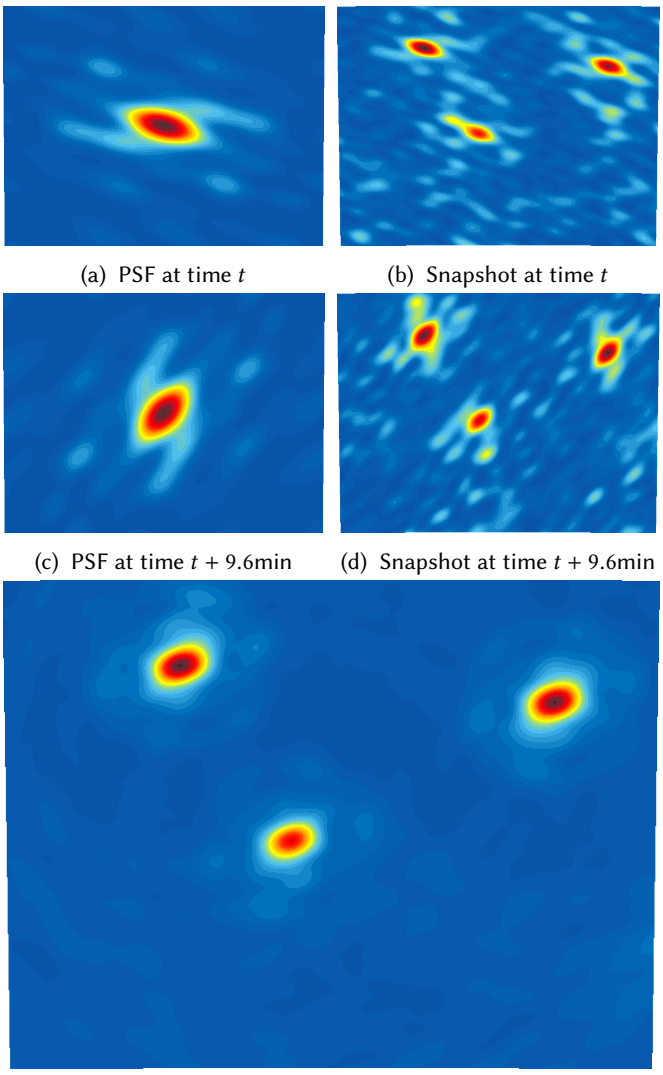


Fig. 16. General form of $W^H \in \mathbb{C}^{M\times L}$ for hierarchical radio-telescopes. Antennas are geographically beamformed together to achieve significant compression. We always have $\|W\|_0 = L$, hence the sparsity is proportional to $\frac{1}{M}$.

sidelobe artefacts. As such, by integrating long enough, noise as well as sidelobes will be averaged out of $I^{\mathcal{T}}_{\widetilde{\mathcal{S}}}$.

Sensitivity Equalization. As seen in Section 2.4, beamforming acts as a spatial filter on the intensity estimate, weighting signals from all directions with the beamshape $\widehat{w}(r) \in \mathbb{C}$. The consequence is that the PSF becomes spatially-varying, and hence the sensitivity of the instrument is non-uniform on the whole field-of-view of interest. If not corrected for, sources far from the center of focus will appear fainter than usual. Moreover, when doing long-exposure imaging, the beamshape itself can act as an artefact, as seen in Figure 18a. To equalize the



(a) PSF at time t (b) Snapshot at time t

(c) PSF at time $t + 9.6$min (d) Snapshot at time $t + 9.6$min

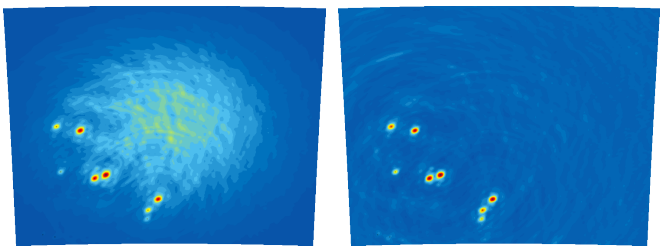
(e) Long-exposure image after 72 snapshots, amounting to 9.6min of observation. All sidelobes have disappeared and the main lobe shrinks due to the rotation. The 3 peaks in the intensity map correspond to the true source locations.

Fig. 17. Effect of Earth's rotation on the PSF. As W changes, the PSF is not time-invariant, but rotates around it's main lobe (17a, 17c). As such, sidelobe artefacts differ at each snapshot (17b, 17d). By integrating sufficiently long enough, the sidelobes combine non-coherently and are averaged out (17e).

sensitivity throughout the field, we can compute the Bluebild estimate of the intensity map when $\widehat{S}(r) \sim \mathcal{N}_c(0, 1)$ in (1). This leads to $\Sigma = W^H \Phi^* \Phi W = G_\Psi$. From here, we can use the standard Bluebild algorithm depicted in Algorithm 2, but as the solutions (D, V) of the generalized eigenvalue problem $\Sigma v_m = \lambda_m G_\Psi v_m$ are $D = V = I_M$, computing the field sensitivity reduces to

$$I_{\text{sensitivity}} = \sum_{m=1}^{M} \left(E_m \circ \overline{E}_m \right),$$

where $E_m = \Phi W_m$. Equalizing a Bluebild estimate $I_{\widetilde{S}}$ then amounts to dividing $I_{\widetilde{S}}$ by I_{snapshot}.



(a) No equalization (b) Equalized

Fig. 18. Intensity estimates $I_{\widetilde{S}}$ of a 7° patch of the sky using LOFAR after 9.6min of observation, with and without equalization. When not equalized (18a), sources distant from the center of the field appear weaker than they should be. Moreover, although there are no sources in the center of the field, the intensity there is nonzero due to the higher sensitivity in the region.

Data anomalies. Assuming a Q point-source model for the sources, we can use (17) to compute the correlation matrix[4] of the data, obtaining

$$\Sigma = \mathbb{E}\left[Y Y^H \right] = \Psi^H \Sigma_F \Psi + \sigma^2 W^H W.$$

Due to the station-level beamforming and assuming the columns of W are normalized to avoid magnifying antenna noise, it can be shown that $W^H W = I_M$, hence only the diagonal entries of Σ are disturbed with antenna-level noise. In practice, antenna visibilities are computed in a dedicated infrastructure using hardware correlators for performance reasons, and due to the above, autocorrelation values are discarded entirely such that Σ is no longer positive-semi-definite. This is problematic as using Σ directly in Bluebild would result in intensity estimates $I_{\widetilde{S}}$ that contain negative amplitudes. We therefore need to correct the data to account for this loss of information.

Two solutions can remedy this problem:

(1) Update Σ by lifting its spectrum (D, V):

$$\widetilde{\Sigma} = V D_{\text{new}} V^H,$$

where $D_{\text{new}} = D - D_{\min}$.

(2) Update Σ by truncating its spectrum (D, V):

$$\widetilde{\Sigma} = V_{\text{new}} D_{\text{new}} V_{\text{new}}^H, \tag{20}$$

where $(D_{\text{new}}, V_{\text{new}})$ is the subset of (D, V) corresponding to positive eigenvalues only.

Both approaches produce a visibility matrix $\widetilde{\Sigma}$ that is positive-semi-definite, hence Bluebild will produce non-negative-valued intensity estimates $I_{\widetilde{S}}$. The drawback of (1) however is that the offset by $D_{\min}$ can be very different from $\text{diag}(\Psi^H \Sigma_F \Psi)$, which is what one would like to shift the spectrum by. In other words, (1) biases the spectrum, making it hard to determine the exact amplitude of point-sources in the sky. On the other hand, (2) is equivalent to truncating the spectrum of the generalized eigenvalue problem $\Sigma v_m = \lambda_m G_\Psi v_m$, hence we opt for this procedure in Bluebild.

Taking all the above into account leads to Algorithm 4 to compute Bluebild intensity estimates in radio-astronomy, and gives us intuition as to why the acoustic optimizations do not hold in this context:

[4] In radio-astronomy jargon, antenna correlation matrices (beamformed or not) are referred to as visibility matrices. We will sometimes use this terminology depending on the context.

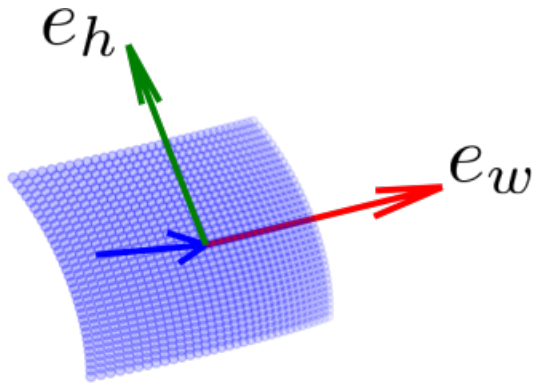


Fig. 19. Basis vectors that span the tangent plane χ at x_{00}. For visualization purposes, e_h and e_w are shown significantly bigger than usual. Their norms coincide with the height/width of the pixels adjacent to the center of the field.

- Φ depends on antenna positions P, and these are no longer time-invariant due to the rotation of the Earth. As such, Φ cannot be computed offline and gets added back to run-time.
- Similar to acoustics, we can compute $\Phi^*\Phi$ offline as it only depends on baselines $\|p_i - p_j\|$ which are rotation invariant, but given the return of Φ to run-time computation, G_Ψ becomes insignificant in comparison.
- Also, although E and $I_{\widetilde{\mathcal{S}}}$ are non-negligeable w.r.t. Φ, their expected run-time is dwarfed by that of Φ.

In light of these observations, what options remain to reduce run-time?

3.3.1 Spatial Synthesis. Given the dominant effect of Φ on run-time and our inability to compute it offline, the only solution to significantly speed up the Bluebild estimate is to compute Φ faster somehow. As seen in Table 4, the bottleneck in computing Φ lies in the evaluation of $N_{\text{px}}L$ complex exponentials. As the floating-point (FP) performance of processors significantly outperforms their transcendental performance, one option is to see if the complex exponential can be approximated using FP operations.

In what follows, assume we use a uniform grid $x_{\text{grid}} \in \mathbb{R}^{N_h \times N_w \times 3}$ as in Figure 4b, defined over a small field-of-view where N_h and N_w represent the number of pixels in the vertical and horizontal directions respectively. Given the above, any point x_{hw} on the grid can be written as

$$x_{hw} = x_{00} + (x_{hw} - x_{00}),$$

where x_{00} is the pixel at the center of the grid. Due to the small field-of-view, $(x_{hw} - x_{00})$ is practically on the tangent plane χ of the sphere at x_{00}, leading to the approximation

$$x_{hw} = x_{00} + (x_{hw} - x_{00}) \approx x_{00} + he_h + we_w,$$

where e_h and e_w are orthogonal vectors on χ aligned with the grid such that $\|e_h\|$ and $\|e_w\|$ equal the height/width of a pixel. (Figure 19.)

Algorithm 4: Long-Exposure-Bluebild($\widehat{\Sigma}$, P, W, τ, λ_c, x_{grid})

Input: $\widehat{\Sigma} = \left\{\widehat{\Sigma}^t \in \mathbb{C}^{M\times M},\ t \in \mathcal{T}\right\}$: visibilities per time-interval
Input: $P = \left\{P^t \in \mathbb{R}^{L\times 3},\ t \in \mathcal{T}\right\}$: antenna positions
Input: $W = \left\{W^t \in \mathbb{C}^{L\times M},\ t \in \mathcal{T}\right\}$: beamforming weights
Input: $\tau \in [0, 1]$: energy threshold
Input: $\lambda_c \in \mathbb{R}$: source signal wavelength
Input: $x_{\text{grid}} \in \mathbb{R}^{N_{\text{px}}\times 3}$: grid-points in Cartesian coordinates
Output: $I^{\mathcal{T}}_{\widetilde{\mathcal{S}}} \in \mathbb{R}^{N_{\text{px}}}$: integrated intensity function

$I^{\mathcal{T}}_{\widetilde{\mathcal{S}}} \leftarrow 0$
for $t \in \mathcal{T}$ **do**
  $I_{\text{snapshot}} \leftarrow$ Bluebild($\widehat{\Sigma}^t, P^t, W^t, \tau, \lambda_c, x_{\text{grid}}$)
  $I^{\mathcal{T}}_{\widetilde{\mathcal{S}}} \leftarrow I^{\mathcal{T}}_{\widetilde{\mathcal{S}}} + I_{\text{snapshot}}$
end
Function *Bluebild*($\widehat{\Sigma}^t, P^t, W^t, \tau, \lambda_c, x_{grid}$)
  Output: $I_{\text{snapshot}} \in \mathbb{R}^{N_{\text{px}}}$: intensity function
  Truncate spectrum of $\widehat{\Sigma}^t$ as in (20), to obtain $\widetilde{\Sigma}^t$
  $G_\Psi \leftarrow W^H\Phi^*\Phi W$, with $\Phi^*\Phi$ as in (13)
  Find M eigenpairs (λ_m, v_m) of the generalized eigenvalue problem

$$\widetilde{\Sigma} v_m = \lambda_m G_\Psi v_m, \qquad \forall m = \{1, \ldots, M\}$$

  Select K leading eigenpairs (λ_m, v_m) such that

$$\tau \leq \sum_{m=1}^{K} \lambda_m \Big/ \sum_{m=1}^{M} \lambda_m$$

  for $m = \{1, \ldots, K\}$ **do**
    $v_m \leftarrow v_m \Big/ \sqrt{v_m^H G_\Psi v_m}$
  end
  $\Phi \leftarrow e^{j\frac{2\pi}{\lambda_c} x_{\text{grid}} P^T}$
  Let $D = [\lambda_1, \ldots, \lambda_K] \in \mathbb{R}^K$ and $V = [v_1, \ldots, v_K] \in \mathbb{C}^{M\times K}$
  $E \leftarrow \Phi W V$
  $I_{\text{snapshot}} \leftarrow (E \circ \overline{E})D$
  Equalize sensitivity on x_{grid}:

$$I_{\text{sensitivity}} \leftarrow \sum_{m=1}^{M} \left(E_m \circ \overline{E}_m\right)$$

  $I_{\text{snapshot}} \leftarrow I_{\text{snapshot}}/I_{\text{sensitivity}}$

Table 4. Run-time breakdown to compute $\Phi \in \mathbb{C}^{N_{\text{px}}\times L}$ for the 1488 HBA antennas of LOFAR. The phase computation is clearly eclipsed by the execution time of the complex exponentials.

	Time (sec)	
N_{px}	e^{jA}	$A = x_{\text{grid}}P^T$
65^2	0.257	0.0292
127^2	1.107	0.128
255^2	3.926	0.423
511^2	17.536	1.687

Developing the expression of $\phi_l(x_{hw})$, we get

$$\begin{aligned}\phi_l(x_{hw}) &= \phi_l(x_{00} + (x_{hw} - x_{00})) \\ &\approx \phi_l(x_{00} + he_h + we_w) \\ &= e^{j\frac{2\pi}{\lambda_c}\langle x_{00}+he_h+we_w, p_l\rangle} \\ &= e^{j\frac{2\pi}{\lambda_c}\langle x_{00}, p_l\rangle}\left(e^{j\frac{2\pi}{\lambda_c}\langle e_h, p_l\rangle}\right)^h\left(e^{j\frac{2\pi}{\lambda_c}\langle e_w, p_l\rangle}\right)^w \\ &= \phi_l(x_{00})\phi_l^h(e_h)\phi_l^w(e_w), \end{aligned} \tag{21}$$

where the approximation on the second line holds due to the continuity of $\phi_l : \mathbb{S}^2 \to \mathbb{C}$. Notice in (21) that regardless of the grid-point x_{hw}, the same three complex exponentials are continuously evaluated. Therefore, computing $\phi_l(x_{00})$, $\phi_l(e_h)$, $\phi_l(e_w)$ exactly once followed by element-wise multiplications is sufficient to compute the synthesis kernel Φ_l over the entire grid. By doing so, we trade $N_h N_w L$ transcendental operations for $N_h N_w L$ floating-point operations, thus removing the previous bottleneck. Algorithm 5 shows an efficient procedure to compute the synthesis kernel $\Phi \in \mathbb{C}^{N_h \times N_w \times L}$, and is graphically illustrated in Figure 20.

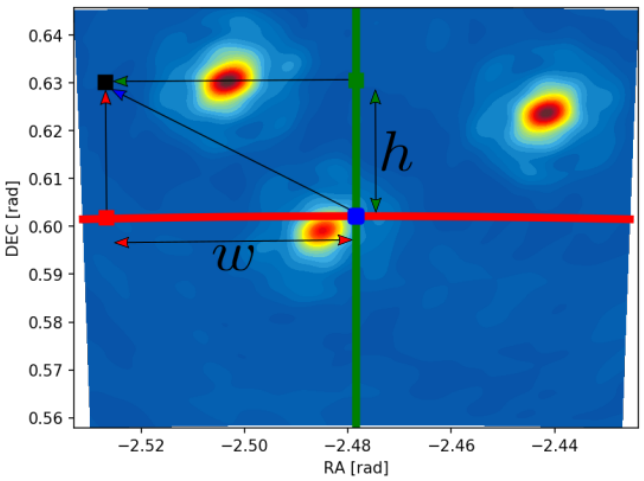


Fig. 20. Geometric application of spatial synthesis: $\phi_l(x_{hw})$ is the product of C, W^w and H^h.

Algorithm 5: compute-$\Phi(P, \lambda_c, x_{\text{grid}})$

Input: $P \in \mathbb{R}^{L\times 3}$: antenna positions
Input: $\lambda_c \in \mathbb{R}$: source signal wavelength
Input: $x_{\text{grid}} \in \mathbb{R}^{N_h \times N_w \times 3}$: grid-points in Cartesian coordinates
Output: $\Phi_{\text{approx}} \in \mathbb{C}^{N_h \times N_w \times L}$: synthesis kernel

Compute grid canonical directions e_h and e_w:

$$e_h \leftarrow x_{\text{grid}}[h/2+1, w/2, :] - x_{\text{grid}}[h/2, w/2, :]$$
$$e_w \leftarrow x_{\text{grid}}[h/2, w/2+1, :] - x_{\text{grid}}[h/2, w/2, :]$$

for $l = \{1, \ldots, L\}$ **do**
  $C_l \leftarrow \phi_l(x_{00})$
  $H_l \leftarrow \left[\phi_l^{-N_h/2}(e_h), \ldots, \phi_l^{N_h/2}(e_h)\right]$ // $H_l \in \mathbb{C}^{N_h}$
  $W_l \leftarrow \left[\phi_l^{-N_w/2}(e_w), \ldots, \phi_l^{N_w/2}(e_w)\right]$ // $W_l \in \mathbb{C}^{N_w}$
  $\Phi_l \leftarrow C_l H_l W_l^T$ // $\Phi_l \in \mathbb{C}^{N_h \times N_w}$
end
$\Phi_{\text{approx}} \leftarrow [\Phi_1, \ldots, \Phi_L] \in \mathbb{C}^{N_h \times N_w \times L}$

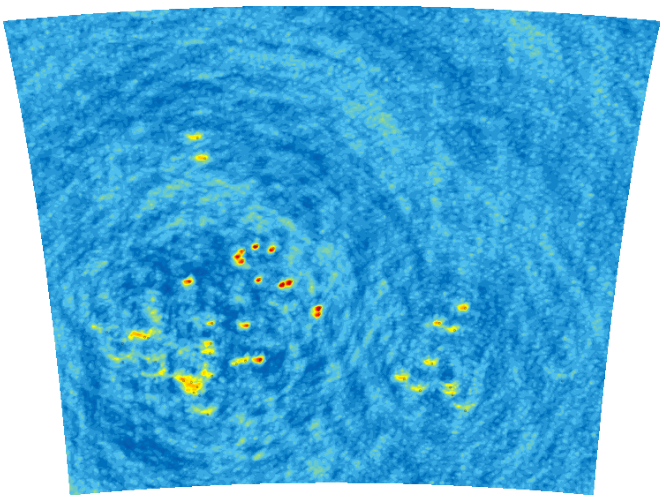
The accuracy of the reconstruction using Φ_{approx} is dependent on the field-of-view being sufficiently small for the sphere to be assimilated to the tangent plane at the center of the field. Therefore, if the field-of-view is large, the small angle approximation no longer holds for pixels located far from the center, hence the image quality is non-uniform throughout the field. This can be seen in Figure 21a where sources in the outer perimeter are significantly blurred compared to the same image computed with Φ_{exact} in 21c. To guarantee a uniform reconstruction over the field, it is therefore necessary to split the imaging grid into regular sub-grids, then use compute-Φ per sub-grid. (Figure 21b) This yields much better estimates with negligeable effects on runtime.

Table 5 shows the performance increase when using Φ_{approx} compared to Φ_{exact}, with an average performance increase of 12x regardless of resolution. Moreover, the performance is actually very dependant on the memory speed of the system. Similar tests on server-class hardware show that the speedup can be up to 20x for larger image sizes.

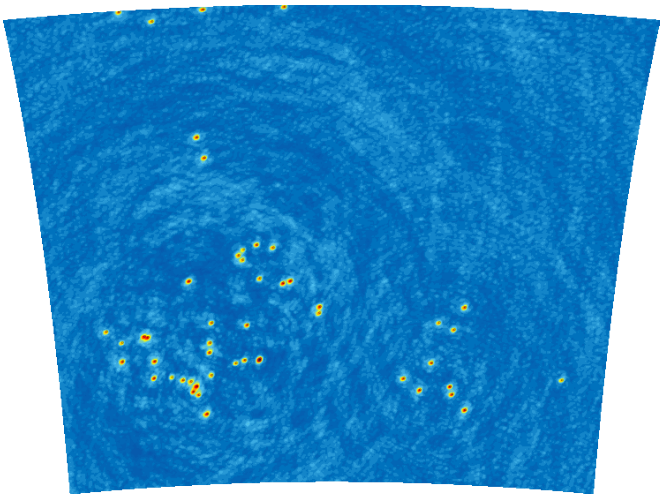
3.3.2 Low-Precision Computing. Notice that $\phi_l(\cdot)$ is always a complex number of modulus 1, hence its real and imaginary components are in the $[-1, 1]$ range. Computing $C_l H_l W_l^T$ element-wise is actually precise enough when computed up to 3 decimals places, and this is guaranteed given the $[-1, 1]$ range restriction. As such, on hardware architectures where one could choose the precision of floating-point operations, a 4x speedup can be achieved by using 16-bit FP units instead of 64-bit units. Unfortunately, this claim cannot be verified for 16-bit arithmetic since NumPy emulates 16-bit operations by masking the least-significant bits and using 32-bit compute units instead. However, Table 6 shows the ~1.7x speed increase when swapping out 64-bit arithmetic for 32-bit arithmetic.

Playing with numerical precision to accelerate the computation of Φ_{exact} is more subtle however. Since $\|p_l\| \geq 10^6$, $\langle r, p_l\rangle$ needs 7 decimals for the integer part. On 32-bit arithmetic already, this is problematic since there will only be 1 decimal digit left to represent the fractional part of $\langle r, p_l\rangle$, which is too coarse and leads to values for Φ_{exact} that are far from the true value. (Figure 22a) Nevertheless, 32-bit arithmetic *can* be used to compute Φ_{exact}, provided that antenna positions all are translated by their center of mass p_0. This transformation has no effect on the final intensity estimate $I_{\widetilde{S}}$ while giving $\|p - p_0\| < 10^5$, hence only 4 decimal digits are needed for the integer part, leaving another 4 for the fractional part to provide sufficient accuracy.

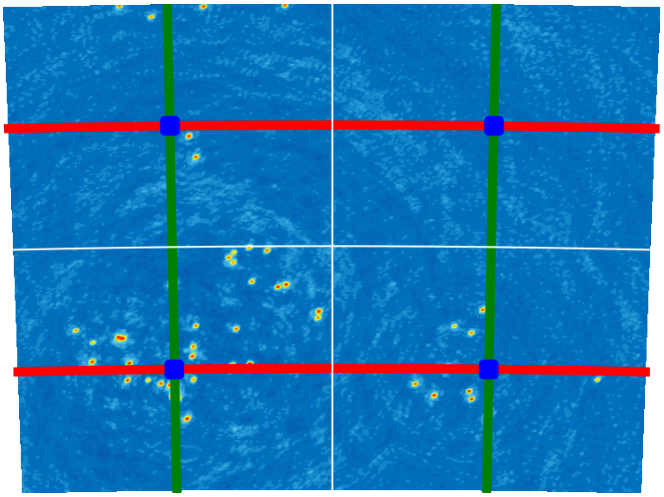
Alternatively, one could always compute $\langle r, p_l\rangle$ in high precision, then do a range reduction step to put the phase in the $[0, 2\pi]$ interval, thus requiring less precision to compute the complex exponentials. However, range reduction is an expensive operation, mitigating any performance benefits from computing the complex exponentials at lower precision.



(a) Spatial synthesis using one large grid covering the entire FoV.



(b) Spatial synthesis using 4 sub-grids overlaid onto 21c.



(c) Exact synthesis using Φ_{exact}.

Fig. 21. Using Bluebild to estimate intensity map $I_{\widetilde{S}}$ of a 19° FoV image after 9.6min of observation using different gridding strategies. In 21a, notice that the four sources at the top of the second quadrant are missing. Moreover, the source clusters in quadrants 3 and 4 are blurred out, hence the grid is too wide given the large FoV. By splitting the grid into four sub-grids, the reconstruction is as good as when using exact synthesis.

Table 5. Run-time comparison between Φ_{exact} and spatial synthesis (Algorithm 5). With at least 12x faster compute speed, spatial synthesis provides an order of magnitude faster image synthesis.

	Time (sec)	
N_{px}	Φ_{exact}	Φ_{approx}
65^2	0.3901	0.00673
127^2	1.5193	0.1196
255^2	6.0757	0.4956
511^2	25.4726	2.2090

Table 6. Effect of numerical precision on run-time. Halving the precision to 32 bits leads to ~1.7x run-time reduction.

	Time (sec)	
N_{px}	$\Phi_{\text{approx-64}}$	$\Phi_{\text{approx-32}}$
65^2	0.00673	0.00393566
127^2	0.1196	0.06838271
255^2	0.4956	0.28833393
511^2	2.2090	1.30458986

3.3.3 Sparse Beamforming. Now that the run-time of Φ is greatly reduced, evaluating E takes up a larger fraction of run-time, making it worthwhile to investigate it further. In acoustics, E was computed as $\Phi(WV)$ because its cost $N_{\text{px}}LM + LM^3$ was inferior to computing E as $(\Phi W)V$ which cost $N_{\text{px}}LM + N_{\text{px}}M^2$. However, as Algorithm 6 shows, the sparsity of W can be used to make the computation of $(\Phi W)V = \Psi V$ much more interesting than $\Phi(WV)$. In effect, Algorithm 6 computes ΦW as M matrix-vector multiplications Ab where $A \in \mathbb{C}^{N_{\text{px}} \times \frac{L}{M}}$, hence the total cost of evaluating Ψ goes down to $N_{\text{px}}L$, bringing the cost of ΨV to $N_{\text{px}}L + N_{\text{px}}M^2$.

In the limit of large image resolutions, we have

$$R = \lim_{N_{\text{px}} \to \infty} \frac{\text{ops}\{\Phi(WV)\}}{\text{ops}\{\Psi V\}} = \frac{LM}{L + M^2} = \frac{M}{1 + M\epsilon},$$

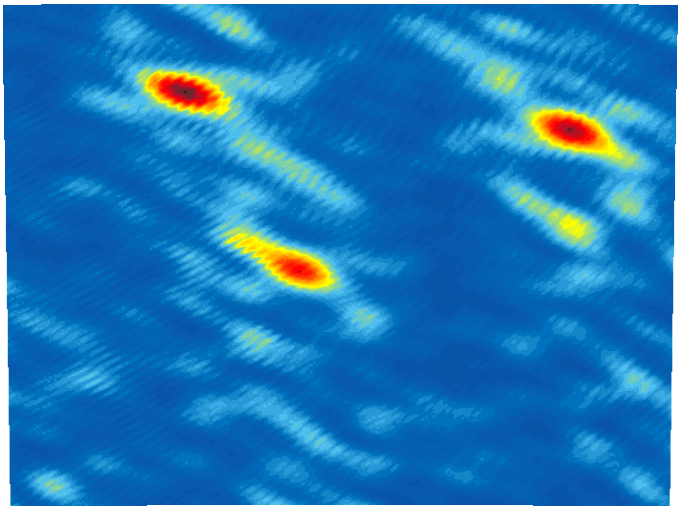
where $\epsilon = \frac{M}{L}$. A simple calculation yields $R > 1$ iff $\epsilon < 1 - \frac{1}{M}$, hence computing ΨV always costs less than $\Phi(WV)$, as is shown for a sample ϵ in Table 7.

Algorithm 6: compute-$\Psi(\Phi, W)$

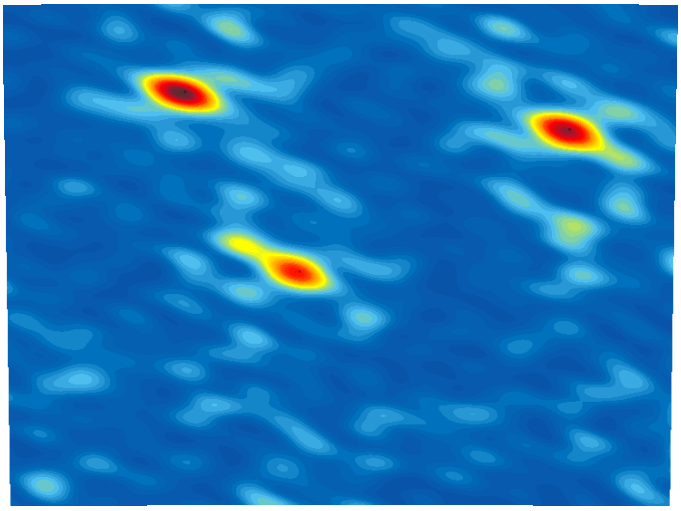
Input: $\Phi \in \mathbb{C}^{N_{\text{px}} \times L}$: synthesis kernel
Input: $W \in \mathbb{C}^{L \times M}$: beamforming weights
Output: $\Psi \in \mathbb{C}^{N_{\text{px}} \times M}$: beamformed synthesis kernel

for $m \in \{1, \ldots, M\}$ **do**
  Let $\Phi_m \in \mathbb{C}^{N_{\text{px}} \times \frac{L}{M}}$ be the synthesis kernels associated to antennas located in station m.
  Let $W_m \in \mathbb{C}^{\frac{L}{M}}$ be the beamforming weights associated to antennas located in station m.
  $\Psi_m \leftarrow \Phi_m W_m$ // $\Psi_m \in \mathbb{C}^{N_{\text{px}}}$
end
$\Psi \leftarrow [\Psi_1, \ldots, \Psi_M]$ // $\Psi \in \mathbb{C}^{N_{\text{px}} \times M}$

3.3.4 Parallelism. Recall from Section 2.2 that one of Bluebild's main highlights is that the intensity estimate $I_{\widetilde{S}}$ is continuously defined. As such, once G_Ψ, D, and V have been computed, each pixel of $I_{\widetilde{S}}$ can be computed independently from all others, and hence we can evaluate pixels in parallel. In practice, we do not compute each pixel individually, but instead partition the imaging grid x_{grid} into sub-grids $\left\{x^p_{\text{grid}},\ p \in \{1, \ldots, \#\text{ partitions}\}\right\}$, then run the sampling step



(a) $I_{\widetilde{S}}$ using exact synthesis and 32-bit precision. Artefacts appear due to insufficient decimals to evaluate the phase-term.



(b) $I_{\widetilde{S}}$ when using spatial synthesis and 16-bit precision. The image is identical to using (exact synthesis, 64-bit) and (exact synthesis, 32-bit, shifted antennas).

Fig. 22. Effect of numerical precision on reconstruction quality.

Table 7. Run-time comparison of sparse Ψ evaluation vs. dense $\Phi(WV)$. Taking sparsity into account always leads to better run-times, regardless of the $\frac{M}{L}$ ratio of the telescope and image size.

	Time (sec)	
N_{px}	ΨV	$\Phi(WV)$
64^2	0.031	0.154
128^2	0.125	0.621
256^2	0.501	2.973
512^2	2.260	11.234

on each sub-grid. Results of Table 8 show a near-linear execution time decrease as more tasks are scheduled in parallel, which is expected since the evaluation of Φ dominates execution time and is now computed in parallel. At the same time, as the data-parallel part of Bluebild is the most time-consuming in practice, using devices like GPUs can lead to significant improvements in execution time (Table 8). It is interesting to note that using optimizations on the GPU result in longer run-times compared to using Algorithm 4 directly. In fact, it is the spatial synthesis step that is causing the slowdown for two reasons:

(1) Trigonometric function throughput is very high on GPUs due to their extensive use in computer graphics. As such, evaluating $e^{j\alpha}$ as $\sin\alpha + j\cos\alpha$ is as fast as doing floating-point arithmetic on most GPUs.
(2) In its current implementation on the GPU, spatial synthesis requires allocating 2 extra arrays to store the synthesis operator Φ on the sub-grid. The constant allocation/deallocation of these arrays adds overhead that becomes noticeable at high resolutions.

In the end, assuming an 8 second integration-time, Table 8 shows that real-time computation of sky images with Bluebild is achievable, both with CPUs (all optimizations combined) and GPUs at a relatively high resolution.

Table 8. Effect of parallel processing on run-time of a 16 MPixel snapshot taken with LOFAR core, assimilated to a $(M, L) = (24, 576)$ radio-telescope in our terminology. We compare the performance of the Intel Xeon E5-2630 CPU to an Nvidia TitanX of the Maxwell generation. "Normal" refers to Algorithm 4, except for the GPU where in addition we use 32-bit numerics (due to limitations of Theano). "RT" refers to Algorithm 4 with spatial synthesis, 32-bit numerics. and sparse beamforming. (We use 32-bit instead of 16-bit since NumPy merely emulates 16-bit computations with 32-bit operations and bit-masking.) Increasing parallelism results in a near-linear decrease in execution time on the CPU. At first glance, the RT optimizations run slower on the GPU compared to using Algo 4 directly.

Device	Algorithm	Time (sec)
E5-2630 – 1 CPU core	Normal	1179.3
E5-2630 – 4 CPU cores	Normal	363.6
E5-2630 – 8 CPU cores	Normal	254.5
E5-2630 – 16 CPU cores	Normal	152.9
E5-2630 – 1 CPU core	RT	97.1
E5-2630 – 4 CPU cores	RT	26.2
E5-2630 – 8 CPU cores	RT	13.4
E5-2630 – 16 CPU cores	RT	7.7
TitanX – 1 GPU	Normal	4
TitanX – 1 GPU	RT	6

4 PYPELINE FRAMEWORK

Early in our quest to achieve real-time imaging with Bluebild in audio and radio-astronomy, the road was frequently blocked by significant barriers hindering our progress:

Telescope variety: the Measurement Set format (`.ms`) is widely used in radio-astronomy to store observation data such as antenna positions, time of experiment and recorded visibilities. However, due to telescope particularities, datasets from different telescopes required a near-complete rewrite of the imaging pipeline each time. Moreover, each Bluebild implementation incorporated

different optimizations such as sparse beamforming or spatial synthesis, making it very difficult to keep track of imaging parameters and quantify improvements in execution time across instruments.

Application proximity: as seen in Section 2, acoustics and radio-astronomy share the same data model, with the main difference being the limitation to sparse beamforming matrices W in radio-astronomy. Therefore, both domains should be able to share the same implementation of Bluebild with minor modifications. This was however not possible as the implementation available at the time was too tailored to radio-astronomy, and hence could not be modified to fit sound field imaging. In an ideal world, the wheel would not have to be reinvented each time: a single general Bluebild implementation should be able to handle all imaging tasks that fit data model 2.1 and instrument model 2.2.

Performance & scalability: the initial version of Bluebild was implemented in MATLAB, but this could not go on indefinitely due to license restrictions, making scalability experiments very difficult. Moreover, optimizations such as spatial synthesis are quite difficult and/or inefficient to implement in MATLAB due to lack of array broadcasting.

Rigidity of the imaging pipeline: the current state-of-the-art imaging pipeline used in radio-astronomy is too tailored to the CLEAN algorithm. As such, the flexibility of Bluebild to generalized beamforming cannot be implemented and tested using pre-existing software packages.

The short story is that both the Bluebild implementation and tools at our disposal at the time lacked the *flexibility* required to properly experiment with the algorithm and associated technologies past small-scale experiments. Therefore, there was a need to design a tool/environment flexible enough to allow algorithms like Bluebild to be implemented while retaining a large degree of freedom for future endeavors. As such, development was heavily shaped by a few principles:

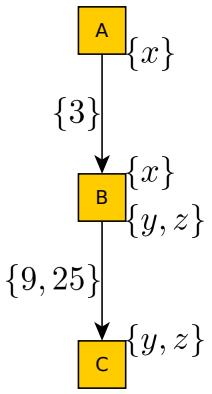
Modularity: make reusable building blocks available to build complex pipelines by module composition.

Resource Efficiency: due to the need to manipulate large arrays efficiently, make maximum use of available compute resources while being lean on memory consumption.

High-Performance & Scalability: cater to small and large imaging tasks.

Ease-of-use: testing of pipeline components and execution of complete pipelines should be easy from the user's point-of-view.

The result of this retooling is *Pypeline*, a Python package to design composable signal processing systems from elementary building blocks. In what follows, we give a brief tour of the main concepts underlying Pypeline, followed by case studies showing the power of the tool for rapid creation and tesing of system prototypes. In what follows, text formatted in monospace font such as `Pypeline` correspond to code concepts or objects.




```
A = A_Spout()
B = B_Bolt()
C = C_Bolt()

A_out = A.execute()
B_out = B.execute(A_out)
C_out = C.execute(B_out)
```

Fig. 23. Sample program made with Pypeline, where elementary `Blocks` are chained together by manually calling their `execute()` functions. Concretely, `A` is a `Spout` that outputs a message containing a unique field `x`. `B` is a `Bolt` that expects messages containing field `x` and outputs a new message with fields `y` and `z`. `C` is a `Bolt` that expects messages containing fields `y`, `z`, and outputs nothing. Instead, it executes a side-effect on the system, such as printing received messages to the terminal.

4.1 Core Concepts

4.1.1 Bolts & Spouts. At its heart, Pypeline is a library based around a message passing paradigm. Concretely, the atomic building block is the `Block`, an object that encapsulates three pieces of information: list of expected inputs, list of expected outputs, and an `execute(msg)` function that does something with received messages. Two types of `Blocks` exist depending on the intended functionality:

`Spouts`: `Blocks` assimilated to producers. `Spouts` don't expect any inputs: every time their `execute(msg=None)` function is called, they produce a value and output it. Possible `Spout` objects include signal generators. Similarly, a `Spout` can act as an abstraction for an acquisition device, outputting acquired samples when calling `execute()`.

`Bolts`: `Blocks` assimilated to consumers. `Bolts` expect a single message as input, and upon calling `execute(msg)` either return a new message or nothing at all. `Bolts` are therefore an abstraction for any function or procedure that takes parameters.

Such abstractions are already very powerful:

- Being the smallest units of Pypeline, `Spouts` and `Bolts` can easily be composed together to form complex processing chains with a high degree of *modularity*.
- Similar to functions, each `Block` in isolation acts like a black box. As far as the user is concerned, all they need to know is the pair (input_specification, output_specification) to correctly use the unit. `Blocks` can therefore implement their functionality however they like. For instance, should a `Bolt` be able to take advantage of specific system resources like GPUs for *high-performance*, then it can do so without affecting the rest of the program.
- Developing new functionality is extremely simple as the user can easily test the correctness of their implementation by manually calling `execute()` as shown in Figure 23.

4.1.2 Topologies. As seen in Figure 23, manually chaining `Blocks` by hand is easy to do and allows simple functional tests. However, manual linking quickly becomes tedious and error-prone when the number of `Blocks` grows since the user has to correctly order the calls to `execute()` to satisfy all dependencies. This can be seen in Figure 24a where `Bolt C` requires messages from `A` and `B` to execute properly. As such, `C.execute()` must be called after `B.execute()` due to its dependency on `B`'s output. To solve this problem, Pypeline allows the `Block` chaining to be specified in the form of a Directed Acyclic Graph (DAG) called a `Topology`. Topologies have the dual role of verifying that the specified node chaining makes sense given the input/output specification of the nodes, and correctly schedule execution of node `execute()` methods.

Recall from Section 4.1.1 that `Bolt` objects must abide by the "one-input, one-output" principle, but as seen in Figure 24a, some `Bolts` require fields found in multiple messages to work correctly. Dealing with such networks therefore requires adding extra logic to handle multiple input/output scenarios. Pypeline solves this problem by introducing a few special `Bolt` objects called `FlowBolts`. The goal of `FlowBolts` are to modify the flow of messages from `Blocks` having multiple inputs or outputs such that their user-facing interfaces do not need to be modified. Two particular `FlowBolts` handle such multi-input (fan-in) and multi-output (fan-out) problems:

`CloneBolts` take a single message stream as input and simply duplicate it to any number of output streams.

`JoinBolts` take multiple message streams as input and return a unique output stream where messages are the union of messages from the input streams.

With these extra abstractions, the action of a `Topology` can be decomposed into 3 stages:

Formal Verification: check the correctness of the network w.r.t. the input/output specifications of each `Block`.

Fan-out Correction: instead of connecting fan-out nodes directly to their descendants, connect the fan-out node's output to a `CloneBolt`, then connect the `CloneBolt` to the descendants. This is illustrated in Figure 24b.

Fan-in Correction: instead of connecting fan-in nodes directly to their ancestors, connect the fan-in node's input to a `JoinBolt`, then route all inbound streams to the `JoinBolt`. This procedure is illustrated in Figure 24c.

At the end of this process, all `Blocks` that are not `FlowBolts` will abide by the "one-input, one-output" principle.

The advantage of this design is three-fold:

- Manual scheduling of `Block` execution is replaced by the simpler `Topology` specification. Moreover, enforcing simple notions like the "one-input, one-output" principle avoid adding complex logic to the user-facing `Block` interface, thus maintaining the system's *ease-of-use* for both large and small networks.
- At a given point in time, nodes that have all their dependencies satisfied can run in parallel using separate processes, thus reducing total execution time. In addition, by using a shared-memory architecture to pass around messages between processes, Pypeline avoids the significant overhead of message serialization/deserialization, further improving execution time of the network while keeping a low memory footprint. Therefore, combining parallel processing and memory-sharing leads to high *resource efficiency*.
- When a particular node B in the network runs slower than its peers, the user can reduce the bottleneck by partitioning its work among several sub-nodes $\{B_1, \ldots, B_N\}$. This can be transparently achieved using a special class of `FlowBolts` called `ScatterBolts` and `GatherBolts` as seen in Figure 25. As such, the system can *scale* to workloads of different size with ease.

4.1.3 Aggregation. As a `Topology` grows in size, the large number of nodes and complex inter-connections between them can make it difficult to grasp the main aspects of a design. In effect, large topologies as depicted in Figure 26 can typically be broken down into a number of sub-components that are connected at a higher level than the `Block`-view implies. As such, to reduce the cognitive load on users and enable a form of self-documentation when reading a design's `Topology` graph, Pypeline allows groups of `Blocks` to be combined together to form `AggregateSpouts` or `AggregateBolts`. Moreover, since aggregates act as standalone `Blocks`, they can be reused as-is in new topologies.

Therefore, aggregation is a powerful tool to abstract away the system's complexity behind different layers, making large designs *easier* to produce and reason about.

4.2 Example: End-to-End Radio-Astronomy Imaging Pipeline with Bluebild

As stated in Section 1, one of the main drawbacks with Bluebild was the rigidity of pre-existing imaging pipelines, making it difficult to create Bluebild images. Using Pypeline, this example shows us to rapidly build an entire imaging pipeline that retains all the flexibility we require.

To keep matters simple, we focus on synthetic data in this example.[5] Given the pre-existing library of components available in Pypeline (Table 9), we simply need to think about the functional components that are required and connect them together. In this case, we need the following types of components:

Data generation: sky model, telescope descriptor, correlation matrix generator, and beamformer.

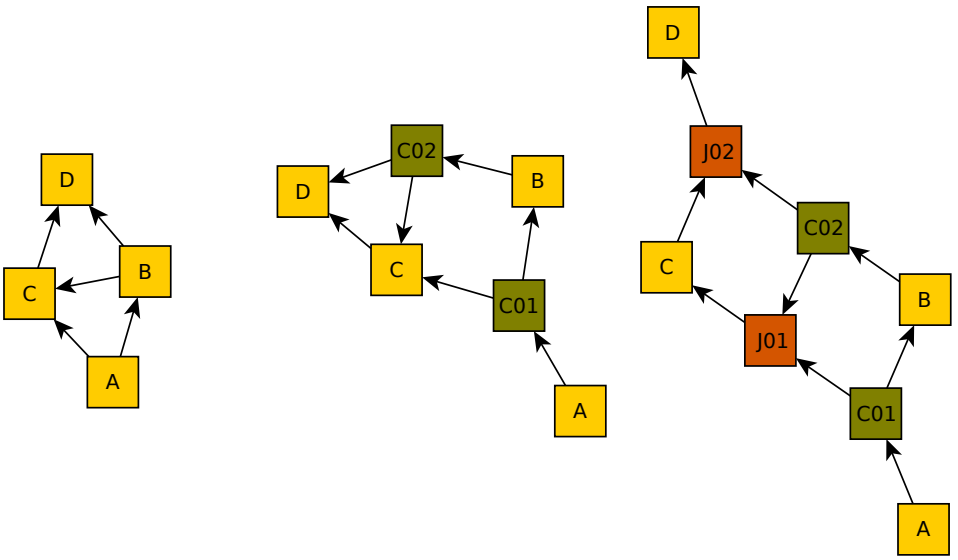
Image generation: Gram model, grid generator, and Bluebild imager.

Image export: buffer (to avoid writing to disk too often), and exporting module.

Once the components have been selected, it is just a matter of connecting them together. Figure 27 shows the network we want, and the code used to obtain it.

4.2.1 Using the GPU. The previous pipeline works flawlessly, but suppose now that you want to generate very high resolution images. To do so, it would be advantageous to use one or more GPUs in the system to accelerate image generation. Assuming the system has four GPUs, minor changes to the graph of Figure 27 suffice to get a massively upgraded system. Concretely, by discarding lines $\{7, 16\}$ and replacing `BlueBildBolt()` on line 8 with `MultiGpuBlueBildBolt`

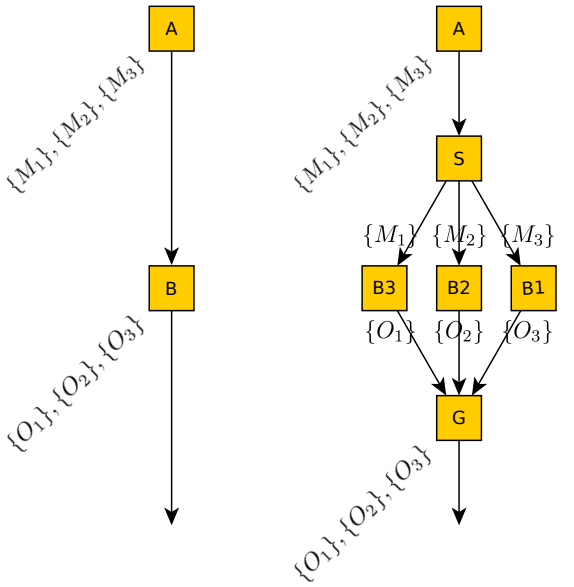
[5] Pypeline already has the ability to handle real datasets and their associated difficulties, namely data cleaning and flag identification.




```
network = Topology(
    nodes={'A': A_Spout(),
           'B': B_Bolt(),
           'C': C_Bolt(),
           'D': D_Bolt()},
    edges={('A','B','C'),
           ('B','C','D'),
           ('C','D')})
network.start()
```

(a) User-specified graph (b) Graph after fan-out correction (c) Graph after fan-in correction

Fig. 24. Sample `Topology` instantiation and transformations. The code describes the intended message flow between nodes and corresponds to network 24a, but violates the "one-input, one-output" principle. Topologies therefore transform the graph in two successive stages: fan-out correction (24b), followed by fan-in correction (24c). At the end of the procedure, all user-specified `Blocks` abide by the "one-input, one-output" principle.



(a) Serial message execution (b) Parallel message execution

Fig. 25. Transparent scaling through workload partitioning. In 25a, `B` is very slow and has a queue of messages $\{M_1\}, \{M_2\}, \{M_3\}$ awaiting processing. As messages are independent, they can be processed in parallel as long as the output messages follow the same order as input messages. In 25b, this is achieved by replicating `B` multiple times, then distributing the workload amongst them using a `ScatterBolt` `S`. Once outgoing messages are computed, a matching `GatherBolt` `G` reorders messages to maintain data consistency.

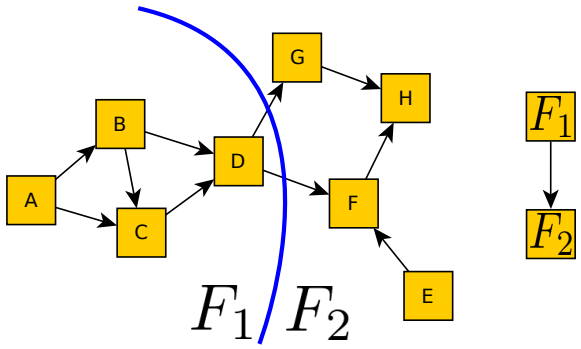


Fig. 26. Large `Topology` designs can often be split into subcomponents. Aggregation therefore allows one to view the system at a different abstraction level.

given in Figure 28, the pipeline transparently makes use of 4 GPUs to form an image. As far as users are concerned, their tool has not changed since the interface of `BlueBildBolt` is respected by `MultiGpuBlueBildBolt`.

4.2.2 Viewing the Bigger Picture. We now have a high-performance imaging pipeline for Bluebild, but the implementation is quite complex due to the myriad of connections between nodes. Before leaving the design at rest, it is advisable to use aggregation to self-document it. One possibility is to split the design into data-generation, image-generation, and export blocks, such that from the top level the design will look like some sampling operator Ψ^* followed by the ideally-matched interpolator $\bar{\Psi}$. This is shown on Figure 29. In addition to being simpler to understand, aggregate blocks can be reused in other designs to save time, thus making development of more complex systems simpler.

5 CONCLUSION

The overiding theme of this thesis was to get close to real-time high resolution interferometric imaging with Bluebild for applications in astronomy and acoustics. As such, the aims of this thesis were:

Table 9. List of `Blocks` already available in Pypeline.

Block	Type	Domain
NoBeamFormerBolt	beamformer	Audio | Radio-Astronomy
FlexiBeamBolt	beamformer	Audio | Radio-Astronomy
DataCleanerBolt	data cleaning	Radio-Astronomy
ExtractedMsSpout	data reader	Radio-Astronomy
NpzWriterBolt	data export	Radio-Astronomy
LaplaceMicrophoneDeviceSpout	device	Audio
GaussianMicrophoneDeviceSpout	device	Audio
RegularCubeMicrophoneDeviceSpout	device	Audio
PyramicMicrophoneDeviceSpout	device	Audio
SphericalMicrophoneDeviceSpout	device	Audio
GramBolt	gram	Audio | Radio-Astronomy
SpectralImageBufferBolt	image buffer	Radio-Astronomy
BinnedSpectralImageBufferBolt	image buffer	Radio-Astronomy
TemporalImageBufferBolt	image buffer	Radio-Astronomy
BscanBolt	imager	Audio | Radio-Astronomy
BlueBildBolt	imager	Audio | Radio-Astronomy
SinoBeamBolt	imager	PET
PsfBolt	imager	Audio | Radio-Astronomy
FilteredBackProjectionImagingBolt	imager	PET
MusicBolt	imager	Audio | Radio-Astronomy
MvdrBolt	imager	Audio | Radio-Astronomy
FocusedGridSpout	imaging grid	Audio | Radio-Astronomy
Full3dSphereGridSpout	imaging grid	Audio
FocusedSphericalGridSpout	imaging grid	Radio-Astronomy
StatisticLevelVisibilityGeneratorBolt	signal generator	Audio | Radio-Astronomy
PositronSignalGeneratorSpout	signal generator	PET
LofarSampleLevelCatalogSkyVisibilityGeneratorSpout	signal generator	Radio-Astronomy
SampleLevelVisibilityGeneratorBolt	signal generator	Radio-Astronomy
LofarStatisticLevelCatalogSkyVisibilityGeneratorSpout	signal generator	Radio-Astronomy
FocusedRandomSkySpout	source generator	Audio | Radio-Astronomy
RandomSkySpout	source generator	Audio | Radio-Astronomy
CatalogSkySpout	source generator	Radio-Astronomy
CircularSkySpout	source generator	Audio
DetailedLiveGuiBolt	visualization	Radio-Astronomy
EigenFuncLiveGuiBolt	visualization	Audio | Radio-Astronomy
LiveGuiBolt	visualization	PET | Audio | Radio-Astronomy

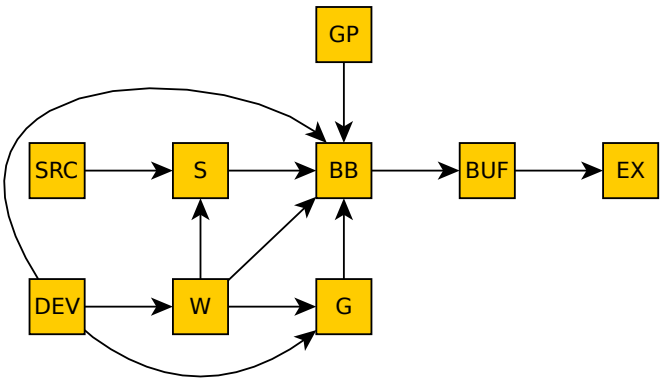
Radio astronomy: Implement the Bluebild algorithm, such that, for reasonable compute power, images could be produced at least as fast as the data is observed. Additionally, we wished to exploit Bluebild's flexibility to produce images for a wide range of beamforming techniques.

Acoustics: Exploiting the common framework with radio astronomy, build a system that produces a sound field from a microphone array. Additionally, we wished to enable a sufficiently fast frame-rate to show the sound field live, and to provide a first comparison of the Bluebild output to that produced by MVDR, B-scan and MUSIC respectively.

Software Development: Develop a modular sensor-array toolkit, that provides the flexibility and speed to test and deploy high-performance algorithms with relative ease.

For radio astronomy, we leveraged algorithm speed-ups and system optimizations to reduce computation by an order of magnitude on CPU architectures, and showed that Bluebild can be efficiently implemented on GPUs by exploiting the inherent parallelism of the sampling stage. We additionally showed the first least-square image estimates using wide-angle beamformers.

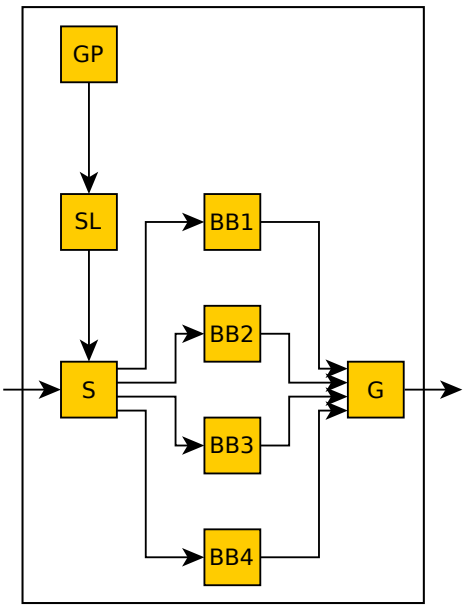
Based on the radio astronomy Bluebild implementation, we built a real-time sound field imager on top of the Pyramic microphone array that accounts for the specifics of acoustic arrays. We showed by experiment that reconstructions of



(a) network

```
network = Topology(
  nodes={'SRC': CatalogSkySpout(...),
         'DEV': ExtractedMsSpout(...),
         'S': StatisticLevelVisibilityGeneratorBolt(...),
         'W': FlexiBeamBolt(...),
         'G': GramBolt(),
         'GP': FocusedGridSpout(...),
         'BB': BlueBildBolt(...),
         'BUF': SpectralImageBufferBolt(...),
         'EX': LiveGuiBolt(...)},
  edges={('SRC','S'),
         ('DEV','BB','W','G'),
         ('S','BB'),
         ('W','S','BB','G'),
         ('G','BB'),
         ('GP','BB'),
         ('BB','BUF'),
         ('BUF','EX')})
network.start()
```

Fig. 27. End-to-end Bluebild simulation pipeline



(a) network

```
MultiGpuBlueBildBolt = AggregateBolt(
    nodes={'GP': FocusedGridSpout(...),
           'SL': SliceBolt(...),
           'S': ScatterBolt(...),
           'G': GatherBolt(...),
           'BB1': BlueBildBolt(...),
           'BB2': BlueBildBolt(...),
           'BB3': BlueBildBolt(...),
           'BB4': BlueBildBolt(...)},
    edges={('IN','S')
           ('GP','SL'),
           ('SL','S'),
           ('S',('BB1','BB2','BB3','BB4')),
           (('BB1','BB2','BB3','BB4'),'G'),
           ('G','OUT')})
```

Fig. 28. GPU-accelerated `BlueBildBolt`. Using a `ScatterBolt` and `GatherBolt`, the grid is cut into pieces and distributed to `BlueBildBolts` $BB_1, \ldots, BB_4$, each of which has control of a separate GPU. Once image chunks are computed, the `GatherBolt` reassembles the final image.

Bluebild seem to be at least on par with those of B-scan and MUSIC, while being more stable than MVDR.

We developed *Pypeline*, a Python package to design composable signal processing systems from elementary building blocks. Through its flexible design, it allowed us to implement entire imaging pipelines for many sensor arrays, allowing algorithms, for example, to be tested with Bluebild (such as Flexibeam and randomised beamforming).

In conclusion, we believe real-time imaging for Bluebild will be achievable, for current and next-generation radio-astronomy instruments, at a reasonable cost. Extrapolating the results of Section 3.3 shows that we could image 400 sub-bands of the LOFAR array at 16 Megapixels using 200 Titan-X (Maxwell) GPUs.

This number will drop substantially. Next generation GPUs (e.g., Tesla P-/V-100s) will bring significant speed-ups, as will hard-coding of certain processing stages in C++, and future algorithm optimizations. For example, the current method recalculates the kernel from scratch, by far the most expensive operation in the chain, at every integration time (of which there can be 3600 in an 8 hour observation window), while the Earth rotation means the kernels are related. Accouting for this redundancy will give another order of magnitude speed-up.

REFERENCES

[1] Orhan Æçal, Paul Hurley, Giovanni Cherubini, and Sanaz Kazemi. 2015. Collaborative randomized beamforming for phased array radio interferometers. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 5654–5658.

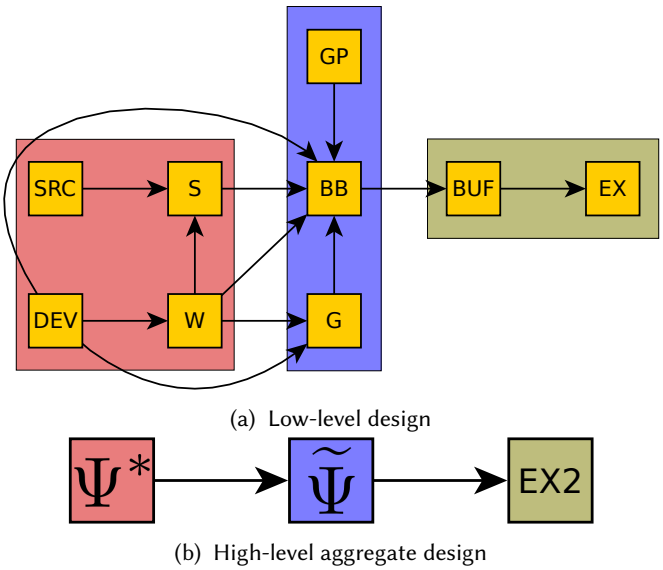
[2] Juan Azcarreta Ortiz. 2016. Pyramic array: An FPGA based platform for many-channel audio acquisition. (2016).

[3] Eric Bezzam, Robin Scheibler, Juan Azcarreta, Hanjie Pan, Matthieu Simeoni, René Beuchat, Paul Hurley, Basile Bruneau, Corentin Ferry, and Sepand Kashani. 2017. Hardware and Software for Reproducible Research in Audio Array Signal Processing. (2017).

[4] Mehrzad Biguesh, Shahrokh Valaee, and Benoît Champagne. 2005. Generalized principal component beamformer for communication systems. *Signal processing* 85, 1 (2005), 67–79.

[5] Basile Bruneau, Eric Bezzam, and Robin Scheibler. [n. d.]. easy-dsp. ([n. d.]). https://lcav.github.io/easy-dsp/

[6] Marco de Vos, Andre W Gunst, and Ronald Nijboer. 2009. The LOFAR telescope: System architecture and signal processing. *Proc. IEEE* 97, 8 (2009), 1431–1437.



(a) Low-level design

(b) High-level aggregate design

Fig. 29. Using aggregation to bring down system complexity.

[7] P Dewdney, W Turner, R Millenaar, R McCool, J Lazio, and T Cornwell. 2013. SKA1 system baseline design. *Document number SKA-TEL-SKO-DD-001 Revision* 1, 1 (2013).

[8] Peter E Dewdney, Peter J Hall, Richard T Schilizzi, and T Joseph LW Lazio. 2009. The square kilometre array. *Proc. IEEE* 97, 8 (2009), 1482–1496.

[9] JA Högbom. 1974. Aperture synthesis with a non-regular distribution of interferometer baselines. *Astronomy and Astrophysics Supplement Series* 15 (1974), 417.

[10] Paul Hurley and Matthieu Simeoni. 2016. Flexibeam: analytic spatial filtering by beamforming. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. Ieee, 2877–2880.

[11] Sepand Kashani, Matthieu Simeoni, and Paul Hurley. [n. d.]. Imaging of Things, Experiments with Bluebild and Sound. ([n. d.]). https://ibm.box.com/s/2z6rc72rv9k773ya74h6e89l6cyt8mdo

[12] Hamid Krim and Mats Viberg. 1996. Two decades of array signal processing research: the parametric approach. *IEEE signal processing magazine* 13, 4 (1996), 67–94.

[13] Hanjie Pan, Robin Scheibler, Eric Bezzam, Ivan Dokmanić, and Martin Vetterli. 2017. FRIDA: FRI-based DOA estimation for arbitrary array layouts. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 3186–3190.

[14] Ralph Schmidt. 1986. Multiple emitter location and signal parameter estimation. *IEEE transactions on antennas and propagation* 34, 3 (1986), 276–280.

[15] Matthieu Martin Jean-Andre Simeoni. 2015. *Towards more accurate and efficient beamformed radio interferometry imaging*. Technical Report.

[16] Matthieu Martin Jean-Andre Simeoni. 2016. *Deconvolution of Gaussian Random Fields using the Graph Fourier Transform*. Technical Report.

[17] Elias M Stein and Guido Weiss. 2016. *Introduction to Fourier analysis on Euclidean spaces (PMS-32)*. Vol. 32. Princeton university press. pg. 154.

[18] Alle-Jan van der Veen and Stefan J Wijnholds. 2013. Signal processing tools for radio astronomy. In *Handbook of Signal Processing Systems*. Springer, 421–463.

[19] Martin Vetterli, Jelena Kovačević, and Vivek K Goyal. 2014. *Foundations of signal processing*. Cambridge University Press.