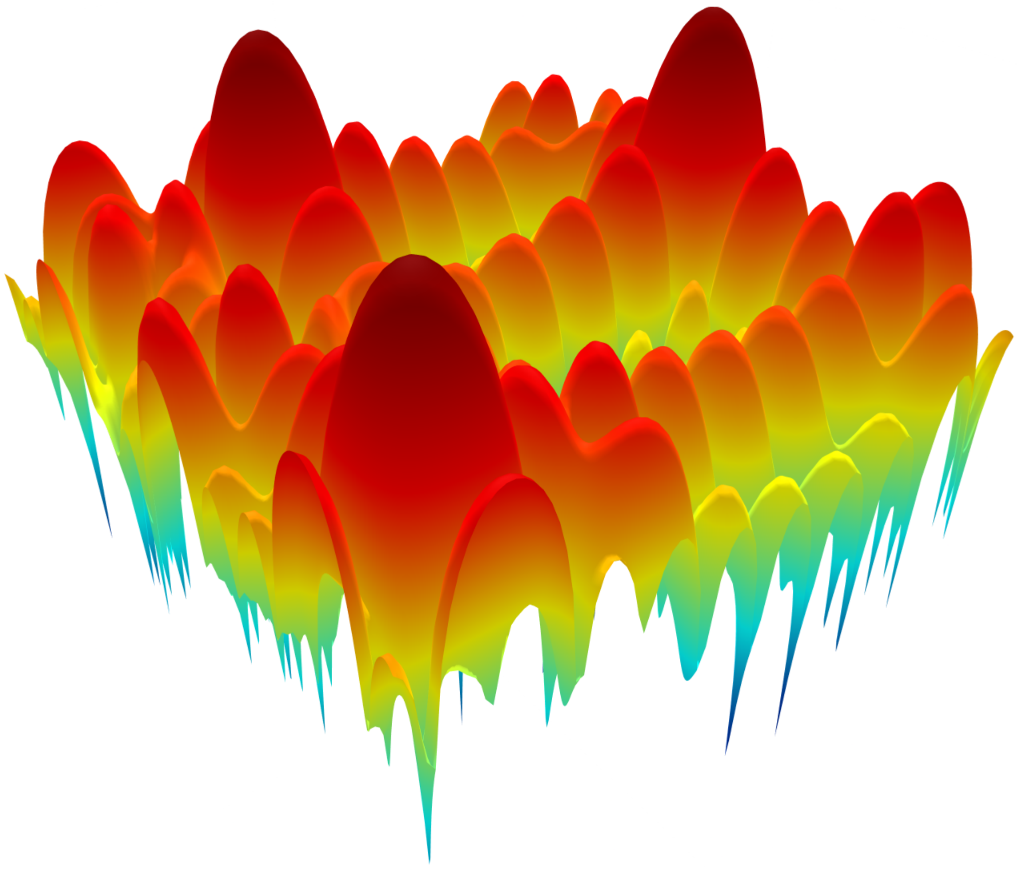
Array Processing ToolBox

gr-array



Author: Rahul Rajeev Pillai

Date: April 8, 2025

Abstract

This is a proposal for a project that seeks to create a tool for designing and analyzing sensor arrays in GNU Radio. The tool will feature a user-friendly GUI to configure array parameters, visualize patterns and experiment with various built-in array processing algorithms. It will also generate GNU Radio (GR) flow graphs based on the user-defined parameters, enabling seamless transition from design to real-time testing withing the GNU Radio. This project strives to make advanced array processing more accessible, fostering exploration and innovation without relying on proprietary software.

Contents

List of Figures

List of Tables

List of Abbreviations

DoA Direction-of-Arrival

GR GNU Radio

GRC GNU Radio Companion

GSoC Google Summer of Code

GUI Graphical User Interface

LCMV Linearly Constrained Minimum Variance

MVDR Minimum Variance Distortionless Response

OOT Out-Of-Tree

UI User Interface

ULA Uniform Linear Array

URA Uniform Rectangular Array

Chapter 1

Introduction

1.1 Project Overview

This project aims to develop an open-source, interactive tool for designing and analyzing sensor arrays within GNU Radio (GR), it is inspired by MATLAB's Phased Array System Toolbox (more specifically: Sensor Array Analyzer). This GUI-based tool should allow users to configure various array parameters, visualize the array-factor in different plots, and play around with popular Beamforming and Direction-of-Arrival (DoA) estimation algorithms.

Once the design is complete, the tool should generate a compatible GR flow graph, configured with the chosen array settings. This functionality bridges the gap between array design and real-time testing, empowering researchers and engineers to iterate and test their designs seamlessly in GR.

By creating this tool, the project seeks to provide the open-source community with a powerful, accessible alternative to proprietary software, lowering the barrier to entry for advanced array processing research and development.

1.2 Motivation

When I first started as an intern, I was tasked with developing adaptive array algorithms. At the time, my understanding of signal processing was very limited, and working with multi-dimensional signal processing algorithms was really a daunting task. The first major hurdle was grasping the mathematical foundations and intuition behind the algorithms. After extensive effort, I was able to develop a good understanding of most concepts.

The next challenge was implementation and verification. While searching for simulation tools, I found that MATLAB was the primary option, but being paid-license based, I was unable to use it. This left me with no choice but to develop everything from scratch. At that time, my programming experience was limited, making the process even more difficult. I looked online for array processing algorithm implementations to get a sense of how they were implemented. Although there were only a handful of resources available, I managed to find a couple of GitHub repositories and PySDR, which proved to be invaluable. They helped me with signal generation, algorithm implementation, pattern visualization, and even understanding the underlying theory. However, they primarily focused on Uniform Linear Array (ULA), making it quite tricky to do the same for planar arrays. Through extensive study of textbooks and careful adaptation of available resources, I was eventually able to extend the simulation to planar array.

While I gained a lot from developing everything from scratch, it was also time-intensive and prone to errors. Developing the core algorithm is one aspect, but implementing signal generation, visualization tools and all the other supporting tools, just adds significant overhead. This project aims to simplify these aspects, allowing researchers and engineers to focus on algorithm development rather than spending excessive time on auxiliary tasks.

1.3 Objective

- **OBJ-1**: To develop an interactive graphical tool that enables a user to configure various parameters of a sensor array and visualize the array factor using different plotting methods.
- **OBJ-2**: The graphical tool should be able to simulate different array processing algorithms for various user-defined array configurations and parameters and provide immediate results.
- **OBJ-3**: The tool should include a mechanism to convert the user-defined configurations into a valid GNU Radio compatible flow graph.

1.4 Milestones

- **MS-1**: Planning
  - Connect with mentors and the community.
  - Iron out details of the project.
  - Setup work environment.
- **MS-2**: Graphical User Interface (GUI).
  - Set up the GUI framework.
  - Implement user-friendly input controls.
  - Ensure smooth navigation and parameter configuration.
- **MS-3**: Signal Processing Library.
  - Implement a function for steering vector generation based on different parameters for a wide range of array configurations.
  - Develop a library of array processing algorithms in Beamforming and DoA estimation.
- **MS-4**: Development of Out-Of-Tree (OOT) modules and flow graph generation.
  - Create a set of customizable GNU Radio Companion (GRC) blocks designed to meet diverse user settings.
  - Implement a functionality within the application window that allows the user to convert their settings to a flow graph in GNU Radio.

- **MS-5**: Integration.
  - Develop a library for generating various types of plots.
  - Integrate the signal processing library, plotting library, GUI and the flowgraph generator with the application window.
  - Enable the window to be launched directly from within GRC.

Table 1.1: Objective - Milestone mapping.

	MS-1	MS-2	MS-3	MS-4	MS-5
OBJ-1	1	1	1		1
OBJ-2	1		1		1
OBJ-3	1			1	1

1.5 Document Structure

The remainder of this proposal is structured as follows:

- **Chapter 2**: Project Systems
  Describes the high-level system architecture. It details how the project is divided into multiple systems components such as the application window, signal processing library, and OOT module and are structured to work together.
- **Chapter 3**: Application Window
  Explains the graphical interface for the project. It gives an idea on how the User Interface (UI) will look like and explains the different panes of the window. It also goes into the different types of plotting methods that are available and how it is connected to the signal processing library.
- **Chapter 4**: Signal Processing Library
  Details the development of core signal processing functions including steering vector generation, beamforming techniques, and DoA estimation algorithms.
- **Chapter 5**: Out-Of-Tree (OOT) Modules
  Describes the implementation of custom GNU Radio OOT blocks that expose array processing functionalities in the GRC environment. This chapter gives a brief introduction to the OOT modules that will be developed for seamless transition between design and testing in GR.

Chapter 2

Project Systems

This entire project can be divided into 3 main components.

- Application Window
- Signal Processing Library
- OOT modules

2.1 Application Window

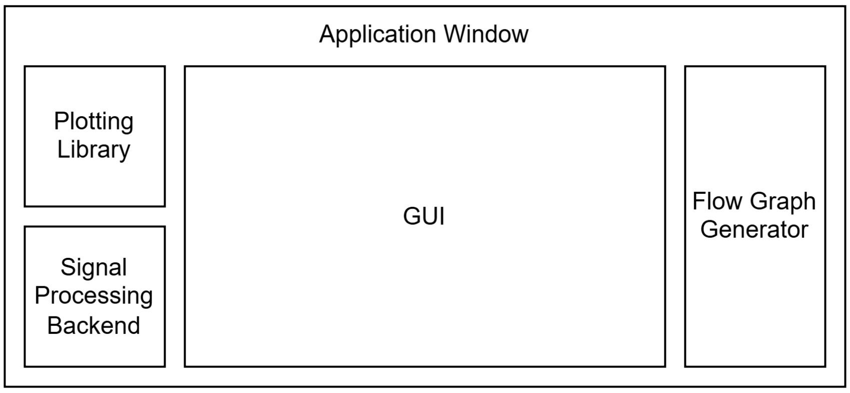


Figure 2.1: Application Window and its sub-systems.

The application window acts as the main interface for the user, bringing together array configuration, visualization, simulation, and flowgraph generation all in a single environment.

It will feature an intuitive GUI with controls like sliders, dropdowns, and text fields to allow users to easily set array parameters, visualize patterns etc. The window will also offer simulation capabilities for array processing algorithms and support automatic generation of flow graphs from user-defined settings that can be used in GNU Radio.

To streamline the workflow, the application window can be launched directly from within GRC, ensuring smooth integration with the existing GNU Radio framework.

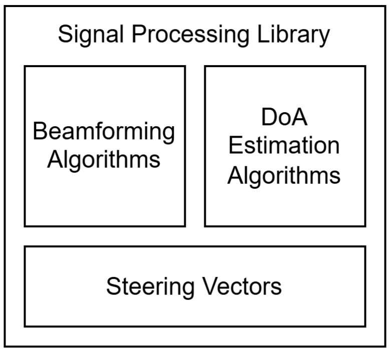


Figure 2.2: Signal Processing Library and its sub-systems.

2.2 Signal Processing Library

The signal processing library forms the core component of the project, providing the computational backbone for array analysis and algorithm simulation.

The library will include functions for steering vector generation, supporting a variety of array geometries and parameter configurations. This ensures flexibility and adaptability across different sensor array setups. Additionally, it will have in-built functions of various Beamforming and DoA estimation, allowing users to evaluate and compare different techniques within the same framework.

2.3 OOT Modules

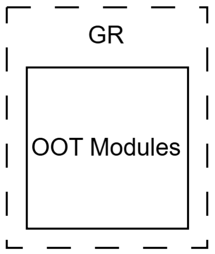


Figure 2.3: OOT Modules.

As part of the system's integration with GNU Radio, a set of custom OOT modules will be developed to extend the platform's native functionality and provide a seamless interface between the graphical tool (application window) and GNU Radio Companion (GRC). These OOT modules are designed to support user-defined array configurations and processing algorithms by using them as reusable blocks within the GRC environment.

Each block will encapsulate a specific array processing function, such as steering vector generation, beamforming, or direction-of-arrival (DoA) estimation, allowing users to incorporate these capabilities directly into their GNU Radio flowgraphs. The blocks will be implemented in C++ or Python, with proper XML code to ensure compatibility with the GRC graphical interface.

This modular design enables users to prototype and validate their array processing pipelines quickly, while maintaining the flexibility to modify and expand the processing chain as needed.

2.4 System Integration

The integration phase brings together all the major components of the project into a functional system. At the heart of this integration lies the seamless interaction between the application window, signal processing library, and the custom OOT modules from GR.

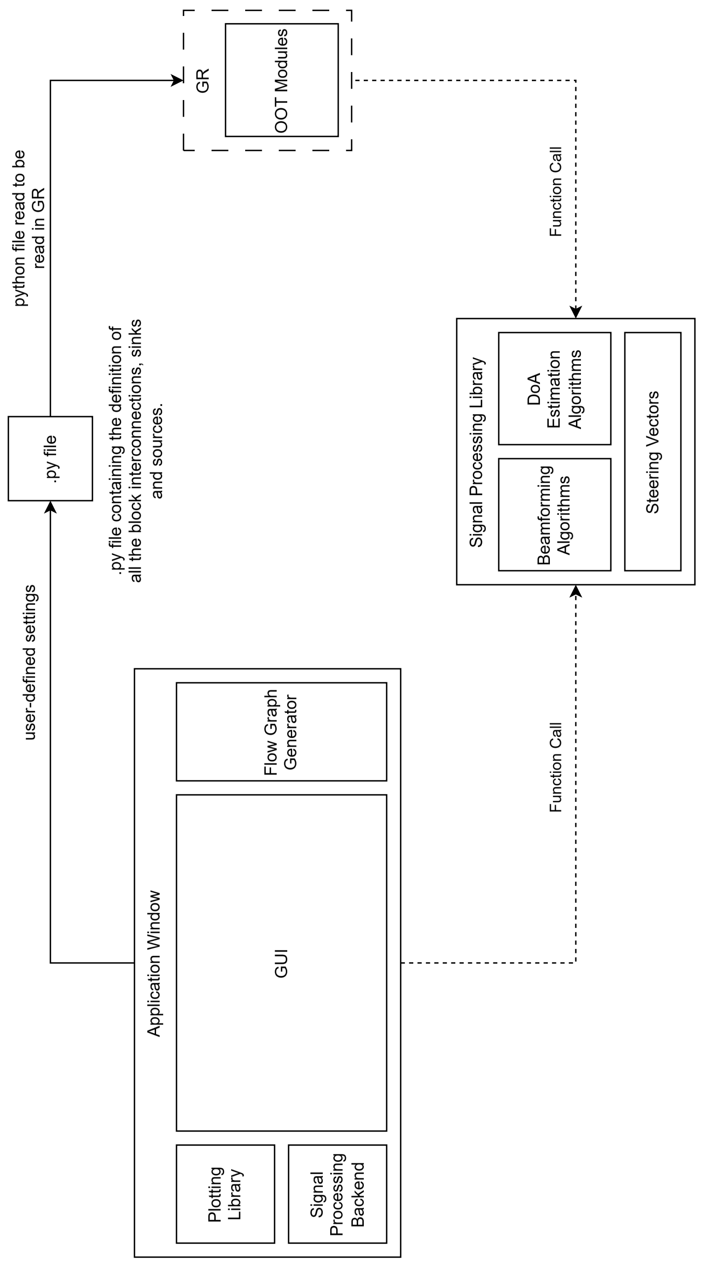

GR
OOT Modules
Function Call
python file read to be read in GR
.py file
.py file containing the definition of all the block interconnections, sinks and sources.
Signal Processing Library
DoA Estimation Algorithms
Beamforming Algorithms
Steering Vectors
user-defined settings
Function Call
Application Window
Flow Graph Generator
GUI
Plotting Library
Signal Processing Backend

Figure 2.4: System block diagram.

The application window serves as the primary user interface, enabling users to intuitively configure sensor array parameters, select array processing algorithms, and visualize the resulting patterns or estimation outputs. It acts as the front end through which all system functionalities are accessed.

Behind the scenes, the signal processing library performs the core computations. It houses essential algorithms for steering vector generation, Beamforming, and DoA estimation. The library is a separate entity common to the application window and the OOT modules and it is accessed through function calls.

To support real-world deployment and SDR experimentation, the system includes a set of custom OOT modules for GR. The application window can generate GR-compatible flow graphs based on the user's settings, enabling a smooth transition from design and simulation to real-world implementation in GR.

Together, these components form a tightly integrated toolchain, from configuration and simulation to real-time execution and bridging the gap between intuitive design and SDR-based prototyping.

Chapter 3

Application Window

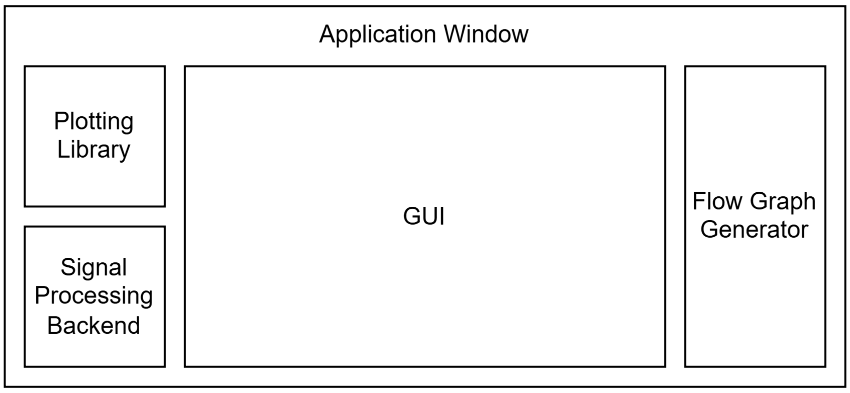


Figure 3.1: Application Window.

The application window is the main container that holds all interface elements and the backend. It provides users with a cohesive environment to configure, run, and analyze the array with various parameters.

3.1 Graphical User Interface (GUI)

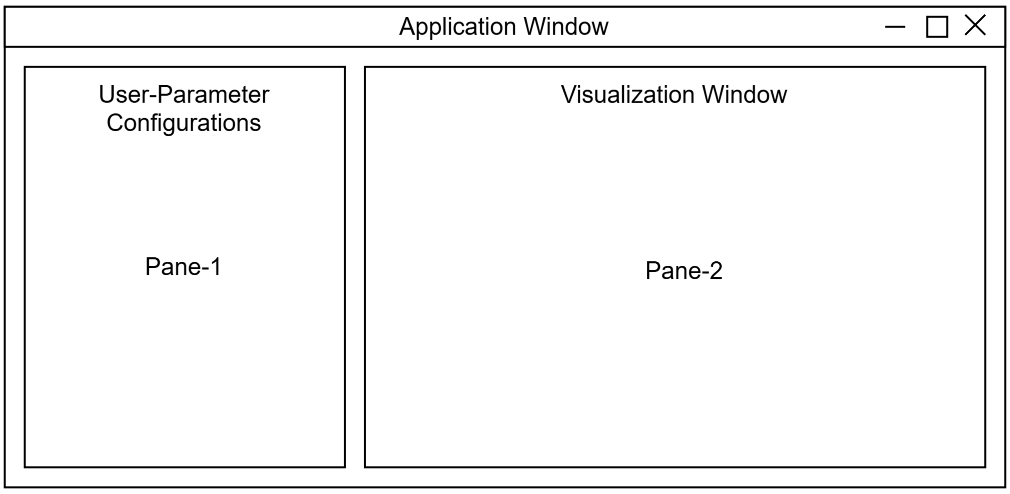


Figure 3.2: Foundational GUI idea for the application window.

The GUI will consists of primarily two panes. Pane-1 will be for the user to input their configurations and settings. And Pane-2 will be for the visualizing the desired plot based on the user-defined settings. Typical parameters available will be,

- Frequency
- inter-element spacing
- number elements
- array configurations
- angle-of arrival

3.2 Signal Processing Backend

This serves as the core engine that connects the user interface with the signal processing library. It receives configuration parameters from the GUI and uses them to initialize or update the processing chain. This backend handles tasks like generating steering vectors, applying beamforming algorithms, and performing DoA estimation using synthetic data. It communicates with the processing library and returns the computed results, which then can be plotted using the plotting library. This architecture ensures a modular responsive, and extensible system.

3.3 Flow Graph Generator

The Flow Graph Generator is a key component that automatically builds and configures a signal processing flow based on user input in GNU Radio. Its main purpose is to translate high-level user-defined parameters into a structured and executable GR flowgraph. This generator acts as a bridge between user-friendly GUI controls and the underlying processing blocks, ensuring that the right sequence of blocks is instantiated and connected in the GNU Radio Companion (GRC).

3.4 Plotting library

The Plotting Library is responsible for rendering all visual outputs in the application. It receives numerical data—like spatial spectra, array responses, or estimated angles—from the signal processing backend and generates clear, interactive plots in the Visualization Window (Pane-2). It includes different plotting methods like,

- Contour plot
- Polar plot
- Cartesian plot

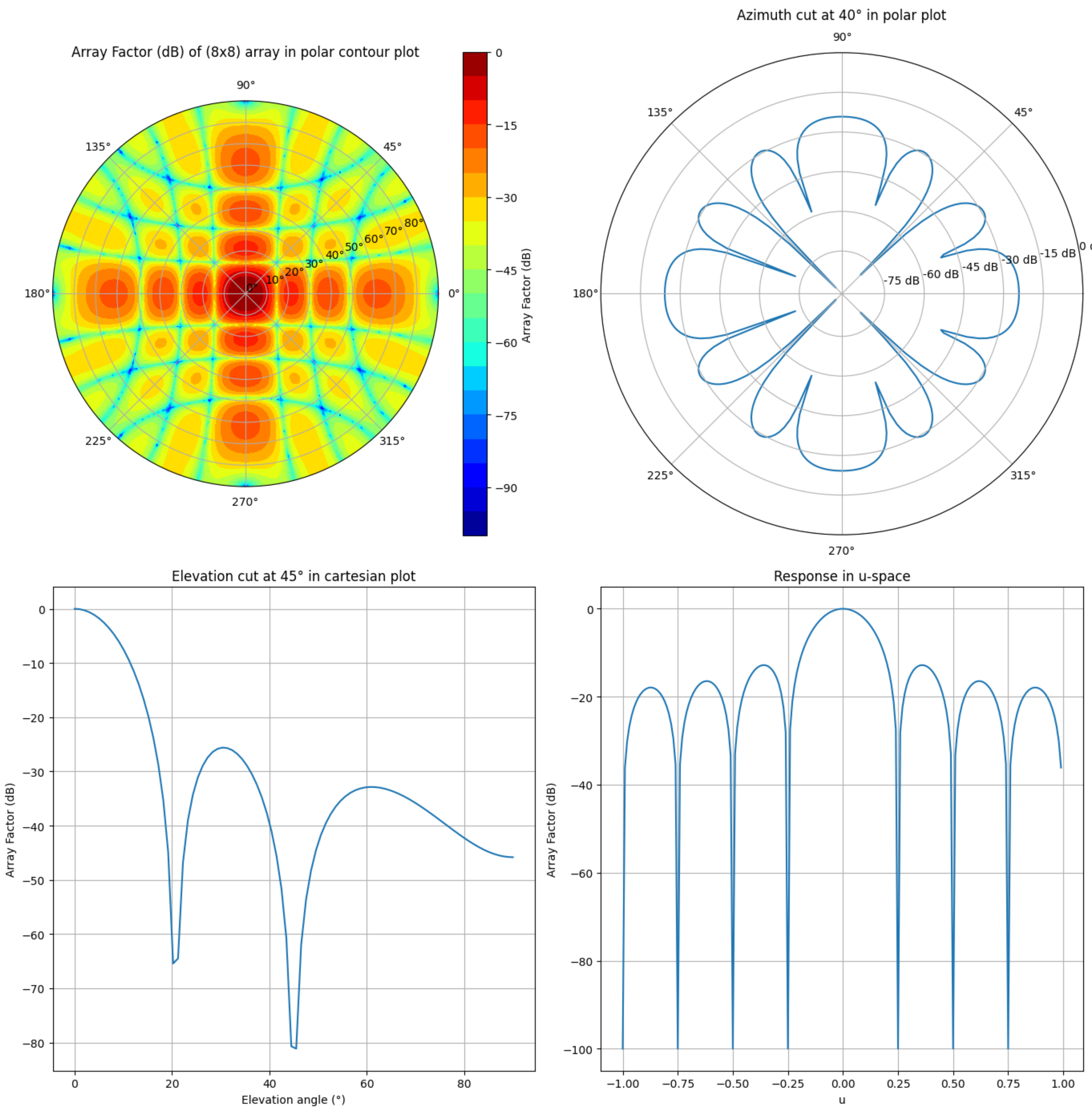


Figure 3.3: Different pattern visualization of an 8×8 array.

Chapter 4

Signal Processing Library

4.1 Array

Table 4.1: Array configuration and its steering vector calculation.

Array	Steering Vector
Uniform Linear Array	$\mathrm{v}(\theta) = e^{-\mathrm{j}2\pi \frac{\mathrm{d}}{\lambda} n \sin(\theta)}$
Uniform Rectangular Array	$\mathrm{v}(\theta,\phi) = e^{-\mathrm{j}2\pi \frac{1}{\lambda}(n\mathrm{d}_x \sin(\theta)\cos(\phi) + m\mathrm{d}_y \sin(\theta)\sin(\phi))}$

where,

- d - inter-element spacing in m.
- λ - wavelength of the signal.
- θ - elevation angle in rad.
- ϕ - azimuth angle in rad.
- d_x - inter-element spacing in x-axis (array normal is z-axis).
- d_y - inter-element spacing in y-axis (array normal is z-axis).
- n - index of the element in x-axis.
- m - index of the element in y-axis.
- $\mathrm{v}(\theta)$, $\mathrm{v}(\theta,\phi)$ - steering vectors for ULA and URA respectively.

4.2 Beamforming algorithm

Table 4.2: Beamforming algorithm and its weights computation.

Algorithm	Weight computation
Minimum Variance Distortionless Response	$\mathrm{w} = \frac{\mathrm{R}^{-1}\mathrm{v}}{\mathrm{v}^{\mathrm{H}}\mathrm{R}^{-1}\mathrm{v}}$
Linearly Constrained Minimum Variance	$\mathrm{w} = \frac{\mathrm{R}^{-1}\mathrm{C}}{\mathrm{C}^{\mathrm{H}}\mathrm{R}^{-1}\mathrm{C}}\delta$

where,

- w - is the weight vector of the algorithm.
- R^{-1} - inverse covariance matrix.
- v - steering vector (either from ULA or URA)
- C - constraint matrix.
$$\mathrm{C} = \left[\mathrm{v}_0, \mathrm{v}_1, \ldots \mathrm{v}_i\right]$$
v_i refers to steering vector based on different directions. C contains all possible steering vector that are to be considered.
- δ - desired response matrix
$$\delta = \left[1, 1, 0, \ldots 1\right]$$
contains the desired response corresponding to the steering vector. 1 means beam and 0 means null.

4.3 DoA Esimation algorihtm

Table 4.3: DoA algorithm and its spectrum computation.

Algorithm	Spectrum computation
Beamscan	$\mathrm{P}(\theta, \phi) = \mathrm{v^H R v}$
Minimum Variance Distortionless Response	$\mathrm{P}(\theta, \phi) = \frac{1}{\mathrm{v^H R^{-1} v}}$

where,

- $\mathrm{P}(\theta, \phi)$ - refers to the power spectrum.

Chapter 5

Out-Of-Tree (OOT) Modules

5.1 Array Response

This GNU Radio block models the effect of a sensor array receiving signals from one or more sources. It takes multiple incoming signals (each representing the signal received at a sensor element), applies phase shifts respective to the element and combines them. The phase shifts are computed based on a steering vector, which encodes the spatial direction of the incoming signal relative to the array geometry. The block outputs N phase-shifted signals, one for each array element, representing the spatially sampled wavefront.

5.2 Steering Vector

This block calculates the complex phase shifts associated with a particular DoA. Given the array geometry, the number of elements, the inter-element spacing, angle of arrival and the frequency, it outputs a vector of phase shifts. This block is used by the Array Response and Beamforming block. This block can either be static (fixed angle) or dynamic (configurable at runtime).

5.3 Beamforming

The beamforming block receives the N signals (already phase-aligned according to the assumed direction of arrival) and applies spatial filtering techniques to enhance signals arriving from a desired direction while suppressing interference and noise from others. It can support different algorithms such as,

- MVDR
- LCMV

5.4 Direction-of-Arrival (DoA) Estimation

This block estimates the directions from which signals are arriving using the N input signals from the array response block and gives the estimated directions as a message. It supports popular algorithms like,

- Beamscan
- MVDR

Chapter 6

Conclusion

6.1 Deliverables

The deliverables will be the 3 components mentioned in the project systems in chapter 2.

- Application Window
  - **DS-1.1**: Plotting Library.
  - **DS-1.2**: Signal Processing backend.
  - **DS-1.3**: GUI.
  - **DS-1.4**: GR flow graph generator.
- Signal Processing Library
  - **DS-2.1**: ULA, URA
  - **DS-2.2**: MVDR, LCMV
  - **DS-2.3**: Beamscan, MVDR
- Out-Of-Tree (OOT) Modules
  - **DS-3.1**: Array Response module
  - **DS-3.2**: Steering Vector module
  - **DS-3.3**: Beamforming module
  - **DS-3.4**: DoA Estimation module

6.2 License

All code created during this project will be released under the GNU General Public License v3 (GPLv3).

6.3 Timeline

This GSoC project schedule spans over 22 weeks, from June-2 to November-10. The minimum guaranteed time spent per week roughly is,

$$3 \text{ hours} \times 6 \text{ days} + 1.5 \text{ hours on sunday} = 19.5 \frac{\text{hours}}{\text{week}}$$

The total time spent on the project,

$$3 \text{ hours} \times 6 \text{ days} \times 22 \text{ weeks} + 1.5 \text{ hours on sunday} \times 22 \text{ weeks} = 429 \text{ hours}$$

I am willing to put additional hours on holidays if necessary.

- **May-8 to June-1**
  - Completion of the milestone MS-1.
- **June-2 to June-16**
  - Development of deliverables from DS-2.1 to DS-2.3.
  - Interactions with mentors regarding the verification of the algorithms.
  - Defining test cases and documentation.
  - Verification.
- **June-17 to July-1**
  - Development of deliverables from DS-3.1 and DS-3.2.
  - Interactions with mentors for the verification of the OOT modules in GR.
  - Defining test cases and documentation.
- **July-2 to July-13**
  - Verification.
  - Presenting the developed algorithms to the mentors and getting feedback.
  - Acting on the feedback.
  - Do pending tasks if any.
  - Completion of the milestone MS-3.
- **Midterm Evaluation Submission - July-14**
  - Submission of deliverables from DS-2.1 to DS-3.2.
  - Test case scripts and related documentation.
  - Verification Report.
- **July-13 to July-27**
  - Development of deliverables from DS-3.3 and DS-3.4.
  - Interactions with mentors for the verification of the OOT modules in GR.

  - Defining test cases and documentation.
  - Verification

- **July-28 to August-3**
  - Conduct research on the Qt Designer.
  - Create a rough sketch of the graphical interface.
  - Get feedback from mentors regarding the GUI.
- **August-4 to August-25**
  - Development of deliverables from DS-1.1 to DS-1.3
  - Completion of milestone MS-2.
  - Meeting with the mentors regarding the developed UI and its functionality.
- **August-26 to September-1**
  - Study about the ways to generate a GNU Radio compatible flow graph from the user-defined parameters.
- **September-2 to September-23**
  - Meeting with mentors regarding the development of DS-1.4 and its test cases.
  - Development of the deliverable DS-1.4.
  - Test cases and documentation.
  - Verification.
  - Completion of milestone MS-4
- **September-24 to September-30**
  - Do any pending tasks or continue with schedule or break.
- **October-1 to October-10**
  - Integration of the plotting library and signal processing backend with the application window.
  - Test cases and documentation.
  - Verification.
- **October-11 to October-31**
  - Integration of the flow graph generator with the application window.
  - Test cases and documentation.
  - Fully integrated verification.
  - Completion of the final milestone MS-5.
- **November-1 to November-9**
  - Demonstration to the mentors.
  - Submission of the project and documentations.

6.4 About me

- Name: Rahul Rajeev Pillai
- Place of residence: Coimbatore, Tamil Nadu, India
- University: Amrita Vishwa Vidyapeetham
- Academic Background: Bachelors in Electrical and Computer Engineering (2024)
- Work Experience: 1 year
- Work Designation: Jr. Design Engineer
- Field: DSP/Communication
- g-mail: rahulpillairj@gmail.com
- github: https://github.com/Ashborn-SM
- Time zone: UTC+5:30

6.5 A note to the mentors

- I am not a student but a full-time employee.
- For the interactions/communications with the mentors, it can be through g-mail, google-meet etc.
- This isn't my first open-source contributions. I started my journey way back in 2020 but had to stop for various reasons. Here's my previous contributions: https://github.com/MakeContributions/DSA
- For demonstrating my coding capabilities, here's the *link* to .ipynb file which contains the code for the various pattern plots seen in chapter 3. The 3-D pattern seen in the title page is the array factor of the weights computed using LCMV, which is also present in that file.
- The first two months of the timeline may look overcrowded but couple of them are already implemented in the .ipynb I shared in the above point. And also, I have experience developing couple of other Beamforming and DoA estimation algorithms in my work. So, in my opinion, it should not be taking any longer than what is projected for the completion of the project.

6.6 Acknowledgement

I've read and understood the GSoC Student Info and the manifest, and I agree to follow the rules, including the three-strike policy. I'll communicate regularly with my mentor, stay transparent about my progress, and follow community guidelines throughout the program.

Cyberspectrum is the best spectrum