ADVANCED ELECTRICAL & COMPUTER ENGINEERING

UNIVERSITY OF KENNESAW

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Project 1: MIMO-GPR Virtual Array Synthesis

Authors:
Student 1 (Hossein Molhem)
Student 2 (ID: your id number)

Date: January 8, 2026

Contents

1 Project Proposal: MIMO-GPR Virtual Array Synthesis

This project serves as the first milestone in bridging Geophysics and Data Science. It will be demonstrated how a dense ”virtual” array can be mathematically generated using a sparse physical antenna array. This technique is considered fundamental to modern high-resolution Ground-Penetrating Radar (GPR) and seismic imaging in 2026.

1. Project Title

Subsurface Resolution Enhancement via MIMO Virtual Array Synthesis

2. Objective

To develop a Python-based simulation that calculates the spatial positions of virtual elements in a MIMO (Multiple-Input Multiple-Output) radar system and visualizes the resulting aperture expansion. This allows for higher angular resolution without increasing the number of physical sensors.

3. Geophysical Context

In GPR, the ability to resolve two closely spaced objects (like two parallel utility pipes) depends on the size of the array’s aperture. Using a MIMO configuration, we create a Virtual Array in which each virtual element represents the effective phase center of a distinct Transmit-Receive (TX-RX) pair.

4. Technical Roadmap & Learning Goals

Phase A: Mathematical Modeling

- **Coordinate Geometry:** Define positions for a set of M transmitters and N receivers (e.g., a ”Minimum Redundancy Array” or a simple linear sparse layout).
- **Synthesis Algorithm:** Implement the vector addition of TX and RX coordinates. The virtual element position for the i-th transmitter and j-th receiver is typically calculated as:

$$POS_{virtual} = \frac{POS_{TX,i} + POS_{RX,j}}{2}$$

- **Redundancy Analysis:** Identify and address overlapping virtual elements that may arise when physical spacing is not optimized.

Phase B: Data Science Integration (Pandas & Seaborn)

- Data Structuring: Store TX, RX, and Virtual positions in a Pandas DataFrame. Use categorical labeling to distinguish antenna types.
- Seaborn Visualization:
  - Use `sns.scatterplot()` to plot the 1D or 2D array layout.
  - Map the `hue` and `style` parameters to the **"Antenna Type"** to clearly visualize the "physical vs. virtual" relationship.
  - Create a **"Density Plot"** of the virtual elements to identify **"effective aperture"** and gaps in the array.

Phase C: Evaluation Metrics

- **Aperture Gain:** Calculate the ratio of the virtual array length to the physical array length.
- **Resolution Prediction:** Estimate the theoretical angular resolution improvement using the formula $\theta \approx \lambda/D$, where D is the new virtual aperture size.

5. Expected Outcomes

- A reusable Python script capable of testing different **MIMO geometries** (e.g., cross-arrays, box arrays, or random sparse arrays).
- High-quality **Seaborn plots** for your portfolio that demonstrate a deep understanding of geophysical signal processing.
- A foundational dataset of virtual positions that will be used in Project 2 (Subsurface Imaging).

6. Required Tools (2026 Stack)

- Python 3.12+
- **Pandas:** For managing complex multi-static pairings.
- **Seaborn/Matplotlib:** For high-fidelity geophysical visualization.
- **NumPy:** For vectorized spatial calculations.

2 Introduction: Major Factors Influencing MIMO System Performance

Understanding the performance of a MIMO system requires more than examining its physical configuration or signal model in isolation. MIMO arrays operate at the intersection of geometry, signal characteristics, hardware limitations, algorithmic choices, and system-level design. Each of these dimensions contributes uniquely to the system's ability to resolve targets, suppress interference, maintain robustness, and operate effectively in real environments.

To develop a complete and accurate picture of MIMO performance, it is essential to organize these influences into coherent categories. This structured perspective not only clarifies how different aspects of the system interact, but also highlights where performance gains can be achieved or where degradation may occur.

In this report, the factors governing MIMO performance are grouped into five major categories:

1. Array Geometry,
2. Signal & Source Characteristics,
3. Hardware & Propagation Effects,
4. Algorithmic Factors, and
5. System-Level Parameters.

Together, these categories form a comprehensive framework for analyzing, comparing, and optimizing MIMO systems. By examining each category in depth, we can better understand the fundamental trade-offs, limitations, and opportunities that shape real-world MIMO performance.

Array Geometry. Array geometry determines the spatial sampling properties of the MIMO system. The physical and virtual apertures, together with the number and distribution of unique virtual elements, define the achievable angular resolution and sidelobe behavior. Element spacing and array topology (such as ULA, sparse, nested, or co-prime structures) further influence aliasing, coarray completeness, and the effective degrees of freedom.

Signal & Source Characteristics. Signal and source characteristics govern how well individual targets or emitters can be separated and estimated. The number of sources relative to the available spatial degrees of freedom, the SNR/INR conditions, and the degree of source coherence strongly affect identifiability and estimation robustness. Bandwidth, number of temporal snapshots, and whether the scenario is far-field or near-field further determine model accuracy and performance limits.

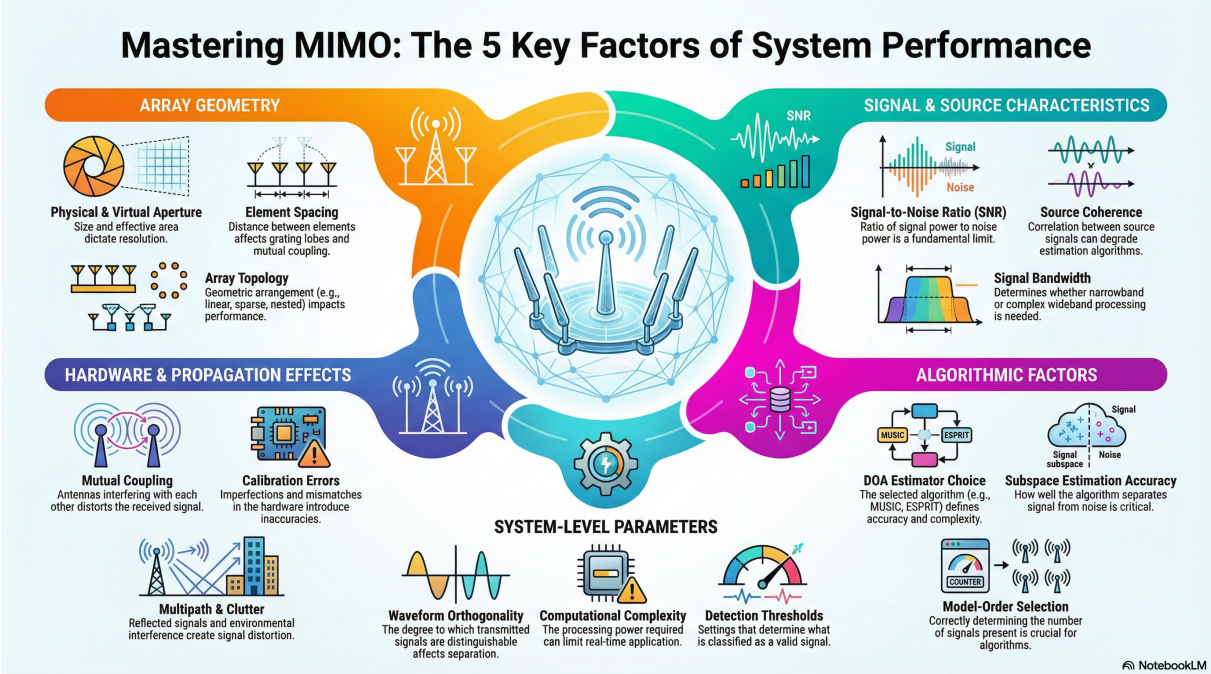

Figure 1

Hardware & Propagation Effects. Hardware imperfections and propagation phenomena create discrepancies between the idealized array model and the real system. Mutual coupling, calibration errors, timing and synchronization offsets, phase noise, and RF impairments can distort the received data and introduce bias. In addition, multipath, clutter, and medium-induced distortions impact the effective channel, often requiring compensation or robust processing techniques.

Algorithmic Factors. Algorithmic choices dictate how efficiently and accurately the available data are converted into parameter estimates. The selection of DOA estimation method (e.g., MUSIC, ESPRIT, ML, or compressive sensing) determines fundamental trade-offs between resolution, robustness, and complexity. Subspace estimation quality, eigenvalue spread, model-order selection, grid design, and regularization or prior information all influence the estimator's bias, variance, and sensitivity to mismatches.

System-Level Parameters. System-level parameters encode how the MIMO configuration is used in practice. Radar mode (monostatic, bistatic, or fully MIMO), waveform orthogonality and coding, PRF, coherent processing interval, and dwell time jointly shape the available diversity and temporal integration. Constraints on computational complexity and latency, together with detection strategies and thresholds (e.g., CFAR or GLRT), ultimately determine the achievable operational performance in realistic scenarios.

Mathematical Foundation of MIMO-GPR Virtual Array Synthesis

3 Core Physical Principle: The Phase Center Theorem

This section explains the physical foundation behind virtual array synthesis. The phase center theorem shows how a bistatic transmitter–receiver pair can be modeled as an equivalent monostatic sensor under far-field conditions, enabling the construction of virtual arrays.

3.1 Bistatic to Monostatic Equivalence

The fundamental concept underpinning virtual array synthesis is the **Effective Phase Center** principle. For a bistatic radar system with transmitter at position $\mathbf{p}_t$ and receiver at position $\mathbf{p}_r$, the round-trip phase to a point target at position $\mathbf{p}_\tau$ is:

$$\phi = \frac{2\pi}{\lambda} \left(\|\mathbf{p}_t - \mathbf{p}_\tau\| + \|\mathbf{p}_r - \mathbf{p}_\tau\| \right) \tag{1}$$

This expression captures the total propagation distance from transmitter to target and back to the receiver.

3.2 Far-Field Approximation

For targets in the **far-field** (Fraunhofer region), where $\|\mathbf{p}_\tau\| \gg \|\mathbf{p}_t\|, \|\mathbf{p}_r\|$, we can approximate:

$$\phi \approx \frac{4\pi}{\lambda} \left\| \frac{\mathbf{p}_t + \mathbf{p}_r}{2} - \mathbf{p}_\tau \right\| \tag{2}$$

In this regime, the bistatic path behaves as if the signal originated from a single effective point, greatly simplifying the geometry.

3.3 Virtual Phase Center

This demonstrates that the bistatic pair behaves as a **monostatic radar** located at the midpoint:

$$\mathbf{p}_v = \frac{\mathbf{p}_t + \mathbf{p}_r}{2} \tag{3}$$

The point $\mathbf{p}_v$ is the **virtual phase center**, and it forms the basis for constructing virtual arrays by combining all transmitter–receiver pairs.

4 Array Geometry Fundamentals

This section introduces the foundational geometric definitions used in MIMO array analysis. The physical array layout determines the available transmitter and receiver positions, while virtual array synthesis expands these into a richer set of spatial sampling points.

4.1 Physical Array Configuration

Let:

- M transmitters at positions: $\mathbf{P}_T = \{\mathbf{p}_{T,1}, \mathbf{p}_{T,2}, \ldots, \mathbf{p}_{T,M}\}$
- N receivers at positions: $\mathbf{P}_R = \{\mathbf{p}_{R,1}, \mathbf{p}_{R,2}, \ldots, \mathbf{p}_{R,N}\}$

The physical array configuration defines the actual hardware layout. These transmitter and receiver coordinates form the basis for constructing the virtual array through pairwise combinations.

4.2 Virtual Array Synthesis

The virtual array $\mathbf{P}_V$ contains $M \times N$ elements:

$$\mathbf{P}_V = \left\{ \mathbf{p}_{V,ij} = \frac{\mathbf{p}_{T,i} + \mathbf{p}_{R,j}}{2} : i = 1, \ldots, M;\ j = 1, \ldots, N \right\} \tag{4}$$

Virtual array synthesis generates a midpoint for every transmitter–receiver pair, effectively expanding the spatial sampling aperture. This virtual geometry enables improved resolution and richer spatial diversity compared to the physical array alone.

5 Linear Array Geometry (1D Case)

This section describes the geometry of one-dimensional linear arrays, where all elements lie along the x-axis. The 1D case forms the foundation for understanding virtual array synthesis and aperture expansion in more complex geometries.

5.1 Linear Array Case (1D)

For a linear array along the x-axis:

- Transmitters: $\mathbf{p}_{T,i} = (x_{T,i}, 0)$
- Receivers: $\mathbf{p}_{R,j} = (x_{R,j}, 0)$

All physical elements share the same vertical coordinate, simplifying the geometry and enabling straightforward virtual position generation.

5.2 Virtual Element Positions

The virtual elements are defined as:

$$x_{V,ij} = \frac{x_{T,i} + x_{R,j}}{2}$$

Each transmitter–receiver pair produces a virtual element located at the midpoint of their x-coordinates. This forms the 1D virtual array used for beamforming and resolution enhancement.

5.3 Aperture Definitions

Physical Array Aperture:

$$D_p = \max(\max(x_T), \max(x_R)) - \min(\min(x_T), \min(x_R)) \tag{5}$$

Virtual Array Aperture:

$$D_v = \max(x_V) - \min(x_V) \tag{6}$$

The physical aperture represents the span of all physical elements, while the virtual aperture reflects the effective span of the synthesized virtual positions. Typically, $D_v > D_p$, enabling improved angular resolution.

5.4 Aperture Expansion Ratio

The aperture expansion ratio is defined as:

$$R = \frac{D_v}{D_p} = \frac{\max(x_V) - \min(x_V)}{\max(\max(x_T), \max(x_R)) - \min(\min(x_T), \min(x_R))} \tag{7}$$

This ratio quantifies how much the virtual array expands the effective aperture relative to the physical array. Larger values of R indicate greater spatial coverage and potential resolution enhancement.

5.5 Aperture Gain

Aperture gain expresses the resolution improvement enabled by virtual aperture synthesis:

$$G = \frac{D_v}{D_p} \tag{8}$$

In uniform linear arrays, aperture gain and aperture expansion ratio are numerically equivalent. However, aperture gain emphasizes the performance benefit (e.g., angular resolution), while expansion ratio reflects geometric scaling. Both metrics are useful for evaluating array efficiency and design trade-offs.

5.6 Aperture Expansion Ratio vs. Aperture Gain

Although the aperture expansion ratio and aperture gain are numerically identical in many linear array configurations, they represent two conceptually distinct quantities.

5.6.1 Aperture Expansion Ratio (Geometry)

The aperture expansion ratio is a **geometric** measure describing how much larger the virtual aperture becomes relative to the physical aperture:

$$R = \frac{D_v}{D_p}. \tag{9}$$

It quantifies the spatial scaling introduced by virtual array synthesis. A value of $R > 1$ indicates that the virtual array spans a larger region than the physical array, enabling finer spatial sampling.

5.6.2 Aperture Gain (Performance)

Aperture gain expresses the **performance improvement** resulting from the expanded aperture. Since angular resolution is inversely proportional to aperture size, the resolution improvement factor is:

$$G = \frac{D_v}{D_p}. \tag{10}$$

Thus, G equals the expansion ratio in linear units, but its interpretation is different: G describes how much the array's resolving capability improves.
When expressed in decibels, aperture gain becomes:

$$G_{\text{dB}} = 10\log_{10}\left(\frac{D_v}{D_p}\right). \tag{11}$$

5.6.3 Conceptual Distinction

- **Aperture Expansion Ratio** describes **how much larger the aperture becomes** (geometry).
- **Aperture Gain** describes **how much the resolution improves** (performance).

Although R and G share the same numerical value in linear units, they serve different roles in array analysis: one characterizes spatial scaling, while the other characterizes the resulting imaging or beamforming benefit.

6 Array Topology Classification

This section summarizes common array topologies used in MIMO systems. Each topology offers different trade-offs in aperture growth, redundancy, and virtual element diversity, making topology selection a key design decision.

6.1 Uniform Linear Array (ULA)

A ULA is the simplest and most widely used array structure, characterized by constant inter-element spacing.

- Equal spacing between adjacent elements
- Spacing typically $d = \lambda/2$
- Virtual array length: $D_V = (M + N - 2)d/2$

The ULA provides a fully contiguous virtual coarray but is more susceptible to mutual coupling and offers limited flexibility in redundancy control.

6.2 Minimum Redundancy Arrays (MRA)

MRAs are designed to minimize repeated virtual positions while maximizing aperture coverage.
For M transmitters and N receivers, the optimal physical spacing minimizes:

$$J = \sum_{x \in \mathcal{X}_V} (r(x) - 1)^2 \tag{12}$$

subject to constraints on total array length.
MRAs achieve the smallest possible redundancy for a given number of elements, producing highly efficient virtual apertures.

6.3 Co-Prime Array

Co-prime arrays exploit number-theoretic spacing to generate large virtual apertures with relatively few physical elements.

- Transmitter spacing: M-element ULA with spacing $N\lambda/2$
- Receiver spacing: N-element ULA with spacing $M\lambda/2$
- Creates $MN - M - N + 1$ unique virtual elements

This structure yields a sparse but hole-free difference coarray, enabling high-resolution processing with reduced hardware complexity.

6.4 Nested Array

Nested arrays combine dense and sparse subarrays to achieve both high resolution and low redundancy.

- Inner ULA with N_1 elements spaced $d_1 = \lambda/2$
- Outer ULA with N_2 elements spaced $d_2 = (N_1 + 1)d_1$
- Creates $N_1N_2 + N_1 + N_2$ virtual elements

The nested structure produces a fully contiguous virtual coarray with significantly fewer physical elements than a full ULA.

7 Redundancy Analysis

This section formalizes how redundancy is quantified in virtual arrays. Redundancy arises when multiple transmitter–receiver pairs generate identical virtual positions, reducing sampling efficiency but also enabling robustness in some configurations.

7.1 Redundancy Matrix

Define the redundancy matrix $\mathbf{R}$ with elements:

$$R_{ij,kl} = \delta(x_{V,ij} - x_{V,kl}) \tag{13}$$

where $\delta(\cdot)$ is the Kronecker delta.
The redundancy matrix encodes pairwise equality between virtual positions. Each entry indicates whether two virtual elements occupy the same spatial location, forming the basis for redundancy quantification.

7.2 Redundancy Count

For virtual position x, the redundancy count is:

$$r(x) = \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{N} \delta(x_{V,ij} - x)\,\delta(x_{V,kl} - x) \tag{14}$$

The redundancy count $r(x)$ measures how many times a specific virtual position is generated across all transmitter–receiver combinations. Higher values indicate repeated sampling of the same location, reducing the number of unique virtual elements.

8 Redundancy Metrics

This section defines two key metrics used to quantify how efficiently a MIMO array converts physical elements into distinct virtual sampling positions. These measures help assess coarray quality, spatial diversity, and geometric efficiency.

Redundancy Ratio:

$$\rho = \frac{\texttt{Total Virtual Elements}}{\texttt{Unique Virtual Elements}} \tag{15}$$

The redundancy ratio ρ measures how many virtual positions are duplicated on average. A value close to 1 indicates minimal redundancy, while larger values imply repeated virtual positions and reduced sampling efficiency.

Array Efficiency:

$$\eta = \frac{\texttt{Unique Virtual Elements}}{\texttt{Physical Elements}} \tag{16}$$

Array efficiency η quantifies how effectively the physical array generates distinct virtual positions. Higher efficiency reflects better geometric expansion and improved spatial resolution potential.

9 2D Array Geometry

This section extends the virtual array formulation to two-dimensional (planar) configurations. By incorporating both x- and y-coordinates, 2D arrays enable improved angular resolution, full azimuth–elevation coverage, and more flexible aperture shaping.

9.1 Planar Array Configuration

For 2D arrays with coordinates (x, y), the virtual array position vector is defined as:

$$\mathbf{p}_{V,ij} = \left(\frac{x_{T,i} + x_{R,j}}{2}, \frac{y_{T,i} + y_{R,j}}{2} \right) \tag{17}$$

This midpoint formulation synthesizes a virtual element for each transmitter–receiver pair, extending the virtual aperture across both spatial dimensions.

9.2 Aperture Area

Physical Array Area:

$$A_P = [\max(x_T, x_R) - \min(x_T, x_R)] \times [\max(y_T, y_R) - \min(y_T, y_R)] \tag{18}$$

Virtual Array Area:

$$A_V = [\max(x_V) - \min(x_V)] \times [\max(y_V) - \min(y_V)] \tag{19}$$

The physical and virtual aperture areas quantify the spatial extent of the array in two dimensions. A larger virtual area generally leads to finer azimuth–elevation resolution and improved imaging capability.

9.3 Point Spread Function

The point spread function (PSF) in the spatial frequency domain (k_x, k_y) is:

$$PSF(k_x, k_y) = \left| \sum_{i=1}^{M} \sum_{j=1}^{N} \exp\left(-j(k_x x_{V,ij} + k_y y_{V,ij})\right) \right|^2 \tag{20}$$

The PSF characterizes the array's 2D spatial response and determines its ability to localize targets in angle. A well-designed virtual geometry yields a narrow, symmetric mainlobe and suppressed sidelobes in both dimensions.

10 Resolution Analysis

This section presents key resolution metrics for physical and virtual arrays, including Rayleigh-based limits, improvement factors, beamwidth expressions, and grating-lobe constraints. These relationships quantify how virtual aperture expansion enhances angular resolution.

10.1 Rayleigh Resolution Criterion

Physical Array Resolution:

$$\Delta\theta_P = \frac{\lambda}{2D_P \cos\theta_0} \tag{21}$$

Virtual Array Resolution:

$$\Delta\theta_V = \frac{\lambda}{2D_V \cos\theta_0} \tag{22}$$

The Rayleigh criterion provides a fundamental limit on angular resolution. Increasing the aperture from D_P to D_V directly reduces the minimum resolvable angle.

10.2 Resolution Improvement Factor

The resolution improvement factor is:

$$F = \frac{\Delta\theta_P}{\Delta\theta_V} = \frac{D_V}{D_P} \tag{23}$$

This factor quantifies how much the virtual array improves resolution relative to the physical array. A larger virtual aperture yields proportionally finer angular discrimination.

10.3 Beamwidth Calculation

The array factor for the virtual array is:

$$AF(\theta) = \sum_{m=1}^{M} \sum_{n=1}^{N} w_{mn} \exp\left(-j \frac{4\pi}{\lambda} x_{V,mn} \sin\theta\right) \tag{24}$$

The half-power beamwidth is approximately:

$$\theta_{BW} \approx 0.886 \frac{\lambda}{2 D_V \cos\theta_0} \quad \text{[radians]} \tag{25}$$

Beamwidth provides a practical measure of angular selectivity. A larger virtual aperture narrows the mainlobe, improving target separation.

10.4 Grating Lobes Condition

Avoid grating lobes when:

$$|d_{\text{eff}}| \leq \frac{\lambda}{2 \sin\theta_{\max}} \tag{26}$$

Here, d_{eff} is the effective spacing between virtual elements. This condition ensures that no secondary lobes appear within the visible region, preserving unambiguous angular estimation.

11 Element Count Relationships

This section summarizes the fundamental relationships between physical elements, total virtual elements, and unique virtual elements in a MIMO array. These relationships help quantify how efficiently the array geometry converts physical hardware into spatial sampling capability.

$$N_V = M \times N \tag{27}$$

The total number of virtual elements N_V is obtained by pairing every transmitter with every receiver. This represents the full set of Tx–Rx combinations before accounting for redundancy.

11.1 Virtual Element Count

For optimal arrays:

$$\left|N_{V,\text{unique}}\right| \approx \frac{MN}{2} \quad \text{(for well-designed arrays)} \tag{28}$$

The number of unique virtual elements reflects how many distinct spatial positions are synthesized. In well-designed sparse or co-prime arrays, approximately half of the total virtual combinations produce unique positions, reducing redundancy while maintaining aperture growth.

11.2 Unique Virtual Element Count

$$\alpha = \frac{N_V}{M+N} \tag{29}$$

The element multiplication factor α measures how many virtual elements are generated per physical element. Higher values indicate more efficient spatial expansion, while lower values suggest redundancy or limited geometric diversity.

12 Geometric Optimization Criteria

This section outlines three core objectives for optimizing virtual array geometry: maximizing aperture span, minimizing redundancy, and maximizing the number of unique virtual positions. These criteria directly impact resolution, sampling efficiency, and coarray completeness.

12.1 Maximize Virtual Aperture

$$\max D_V = \max(x_V) - \min(x_V) \tag{30}$$

$$\text{subject to:} \quad x_{T,i} \in [0, L_T], \quad x_{R,j} \in [0, L_R] \tag{31}$$

Maximizing the virtual aperture D_V increases angular resolution and spatial coverage. The constraint ensures transmitter and receiver positions remain within their physical bounds.

12.2 Minimize Redundancy

$$\min \sum_{x \in \mathcal{X}_V} (r(x) - 1)^2 \tag{32}$$

This objective penalizes repeated virtual positions, promoting a more uniform and efficient sampling of the aperture. The redundancy count $r(x)$ reflects how many times each virtual position is synthesized.

12.3 Maximize Unique Virtual Elements

$$\max |\mathcal{X}_V| \tag{33}$$

Maximizing the cardinality of $\mathcal{X}_V$, the set of unique virtual positions, enhances spatial diversity and coarray completeness. A larger unique set improves resolution and reduces ambiguity.

13 Array Performance Metrics

This section defines key metrics used to evaluate the spatial efficiency and sampling quality of MIMO arrays. These metrics quantify geometric expansion, sampling density, and distribution uniformity of virtual elements.

13.1 Geometric Gain

$$G_{\text{geom}} = \frac{D_V}{D_P} \times \frac{N_{V,\text{unique}}}{M+N} \tag{34}$$

Geometric gain measures the normalized aperture expansion and virtual sampling efficiency relative to the physical array. Here, D_V and D_P denote the virtual and physical aperture spans, respectively, and $N_{V,\text{unique}}$ is the number of unique virtual elements.

13.2 Fill Factor

$$F = \frac{N_{V,\text{unique}}}{D_V/d_{\min}} \tag{35}$$

The fill factor quantifies how densely the virtual aperture is populated relative to the minimum required spacing $d_{\min}$. A value close to 1 indicates efficient spatial utilization without excessive redundancy.

13.3 Uniformity Index

$$U = 1 - \frac{\sigma_d}{\bar{d}} \tag{36}$$

The uniformity index captures the regularity of virtual element spacing. Here, σ_d is the standard deviation and $\bar{d}$ is the mean spacing between adjacent virtual elements. A value near 1 indicates highly uniform spacing, while lower values suggest irregular or clustered distributions.

14 Implementation Considerations

This section outlines practical aspects of implementing virtual array systems, including computational complexity, numerical precision, and memory requirements.

14.1 Computational Complexity

- Virtual array generation: $O(MN)$
- Redundancy analysis (naive): $O((MN)^2)$
- Optimized algorithms: $O(MN \log(MN))$

These complexities reflect the cost of generating virtual positions and analyzing redundancy. Efficient implementations can significantly reduce runtime for large arrays.

14.2 Numerical Precision

$$\phi_{ij} = \mod\left(\frac{4\pi}{\lambda} x_{V,ij} \sin\theta,\ 2\pi\right) \tag{37}$$

Phase calculations require careful handling of modulo operations to ensure numerical stability, especially when dealing with high-resolution virtual positions and small wavelengths.

14.3 Memory Requirements

$$\text{Memory} \approx 8 \times M \times N \quad \text{bytes (double precision)} \tag{38}$$

Memory usage scales linearly with the number of virtual elements. For double-precision storage, each element requires 8 bytes, making memory efficiency critical for large-scale systems.

15 Problem Statement 1: Design and Performance Analysis of a Staggered 2x2 MIMO ULA

1. Context

A compact Ground Penetrating Radar (GPR) system is being designed with a center frequency of **500 MHz**. The antenna system utilizes a **2x2 MIMO** (Multiple-Input Multiple-Output) architecture arranged in a staggered Uniform Linear Array (ULA) topology.

2. Array Constraints

The physical array is constrained by specific sensor placements relative to the wavelength (λ):

- **Transmitters ($M = 2$):** Positioned at 0 and $\lambda/2$.
- **Receivers ($N = 2$):** Positioned at $\lambda/4$ and $3\lambda/4$.

3. Required Tasks

Based on the defined topology, perform the following numerical analyses:

1. **Virtual Array Synthesis:** Calculate the spatial coordinates of all resulting virtual elements using the phase center approximation.
2. **Redundancy Check:** Identify any spatial overlaps in the virtual domain and determine the **Array Efficiency** (η) using the formula:

$$\eta = \frac{\text{Number of Unique Virtual Elements}}{\text{Total Number of Virtual Elements}}$$

3. **Aperture Analysis:**
   - Calculate the **Physical Aperture** width (D_{phys}).
   - Calculate the synthesized **Virtual Aperture** width (D_{virt}).
   - Determine the **Aperture Expansion Ratio**.
4. **Resolution Assessment:** Determine the theoretical angular resolution (θ) for both the physical and virtual arrays to evaluate if the MIMO configuration provides any resolution gain.

Solution: Analysis of Basic 2x2 Staggered ULA

1. System Parameters

Given the operating frequency $f = 500$ MHz, we calculate the wavelength λ in free space:

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8 \text{ m/s}}{500 \times 10^6 \text{ Hz}} = 0.6 \text{ meters} \tag{39}$$

The standard element spacing is defined as $d = \lambda/2 = 0.3$ meters.
The physical sensor coordinates are defined as:

- **Transmitters (T_x):** $T = \{0, d\} = \{0.00, 0.30\}$ m

- **Receivers (R_x):** $R = \{d/2, 3d/2\} = \{0.15, 0.45\}$ m

```
# Element Spacing in meters
d = 0.3  # Element spacing (\lambda/2)

# Given physical array configuration
TX_positions = [0.0, d]  # Transmitter positions in meters
RX_positions = [d/2, 3*d/2]  # Receiver positions in meters

print("="*70)
print("EXAMPLE 1: BASIC 2\times 2 MIMO ARRAY")
print("="*70)

print("\n1. PHYSICAL ARRAY CONFIGURATION:")
print(f"   Transmitters (M=2): TX_1 = {TX_positions[0]:.2f}m,
   TX_1 = {TX_positions[1]:.2f}m")
print(f"   Receivers (N=2):    RX_1 = {RX_positions[0]:.2f}m,
   RX_1 = {RX_positions[1]:.2f}m")

print("\n2. CALCULATING VIRTUAL POSITIONS (Midpoints):")
print("   Formula: Virtual = (TX + RX) / 2")
```

Listing 1: Element Spacing in meters

Output:

```
============================================================
EXAMPLE 1: BASIC 2x2 MIMO ARRAY
============================================================

1. PHYSICAL ARRAY CONFIGURATION:
   Transmitters (M=2): TX1 = 0.00m, TX2 = 0.30m
   Receivers (N=2):    RX1 = 0.15m, RX2 = 0.45m

2. CALCULATING VIRTUAL POSITIONS (Midpoints):
   Formula: Virtual = (TX + RX) / 2
```

2. Virtual Array Synthesis

The virtual element positions V are derived using the convolution (midpoint) formula:

$$V_{mn} = \frac{T_m + R_n}{2}$$

Calculating for all $M \times N = 4$ pairs:

$$V_{11} = \frac{0.00 + 0.15}{2} = 0.075 \text{ m}$$

$$V_{12} = \frac{0.00 + 0.45}{2} = 0.225 \text{ m}$$
$$V_{21} = \frac{0.30 + 0.15}{2} = 0.225 \text{ m} \quad (\textit{Overlap})$$
$$V_{22} = \frac{0.30 + 0.45}{2} = 0.375 \text{ m}$$

The resulting set of virtual positions is:

$$V = \{0.075, 0.225, 0.225, 0.375\} \text{ m}$$

```
# virtual array synthesis

virtual_positions = []

print("\n   Pair 1: TX_1-RX_1")
print(f"   V_1 = (TX_1 + RX_1)/2 = ({TX_positions[0]} + {
   RX_positions[0]})/2 = {TX_positions[0]+RX_positions[0]}/2 = {(
   TX_positions[0]+RX_positions[0])/2}m")
V1 = (TX_positions[0] + RX_positions[0]) / 2
virtual_positions.append(V1)

print("\n Pair 2: TX_1-RX_2")
print(f"   V_2 = (TX_1 + RX_1)/2 = ({TX_positions[0]:.3f} + {
   RX_positions[1]:.3f})/2 = {TX_positions[0]+RX_positions[1]:.3f
   }/2 = {(TX_positions[0]+RX_positions[1])/2:.3f}m")
V2 = (TX_positions[0] + RX_positions[1]) / 2
virtual_positions.append(V2)

print("\n   Pair 3: TX_2-RX_1")
print(f"   V_3 = (TX_2 + RX_1)/2 = ({TX_positions[1]:.3f} + {
   RX_positions[0]:.3f})/2 = {TX_positions[1]+RX_positions[0]:.3f
   }/2 = {(TX_positions[1]+RX_positions[0])/2:.3f}m")
V3 = (TX_positions[1] + RX_positions[0]) / 2
virtual_positions.append(V3)

print("\n   Pair 4: TX_2-RX_2")
print(f"   V_4 = (TX_2 + RX_2)/2 = ({TX_positions[1]:.3f} + {
   RX_positions[1]:.3f})/2 = {TX_positions[1]+RX_positions[1]:.3f
   }/2 = {(TX_positions[1]+RX_positions[1])/2:.3f}m")
V4 = (TX_positions[1] + RX_positions[1]) / 2
virtual_positions.append(V4)

print(f"\n   All Virtual Positions: {[f'{pos:.3f}' for pos in
   virtual_positions]}m")
```

Listing 2: Virtual Array Synthesis

Output:

```
Pair 1: TX_1-RX_1
V_1 = (TX_1 + RX_1)/2 = (0.0 + 0.15)/2 = 0.15/2 = 0.075m

Pair 2: TX_1-RX_2
V_2 = (TX_1 + RX_2)/2 = (0.000 + 0.450)/2 = 0.450/2 = 0.225m

Pair 3: TX_2-RX_1
V_3 = (TX_2 + RX_1)/2 = (0.300 + 0.150)/2 = 0.450/2 = 0.225m

Pair 4: TX_2-RX_2
V_4 = (TX_2 + RX_2)/2 = (0.300 + 0.450)/2 = 0.750/2 = 0.375m

All Virtual Positions: ['0.075', '0.225', '0.225', '0.375']m
```

Interpretation: The virtual array synthesis results confirm that the midpoint operation generates four virtual positions from the 2×2 Tx–Rx pairing. However, two of these positions coincide at $0.225\,\text{m}$, indicating an overlap between the (T_1, R_2) and (T_2, R_1) combinations. This duplication reveals that the virtual array does not fully exploit the available Tx–Rx geometry to create four distinct spatial samples. Instead, the resulting virtual aperture contains only three unique positions, which limits the effective spatial diversity and reduces the potential benefits typically expected from MIMO virtual array expansion.

3. Aperture Analysis

The aperture width is defined as the distance between the maximum and minimum sensor positions ($P_{max} - P_{min}$).

A. Physical Aperture (D_{phys}) Considering the convex hull of all physical sensors ($T \cup R$):

$$P_{phys} = \{0.00, 0.15, 0.30, 0.45\}$$

$$D_{phys} = 0.45 - 0.00 = 0.450 \text{ m}$$

B. Virtual Aperture (D_{virt})

$$D_{virt} = \max(V) - \min(V) = 0.375 - 0.075 = 0.300 \text{ m}$$

C. Aperture Expansion Ratio

$$\text{Gain} = \frac{D_{virt}}{D_{phys}} = \frac{0.300}{0.450} = 0.667$$

Observation: The virtual aperture is actually smaller than the physical aperture. This indicates a sub-optimal geometric arrangement where the phase centers are concentrated in the middle of the array rather than expanding the boundaries.

```
# physical aperture:
combined_positions = TX_positions + RX_positions # combine TX and
    RX positions
physical_aperture = max(combined_positions) - min(
   combined_positions) # physical aperture
print(f"   Physical Aperture: {physical_aperture:.3f}m")

# virtual aperture:
virtual_aperture = max(virtual_positions) - min(virtual_positions
   ) # virtual aperture
print(f"   Virtual Aperture: {virtual_aperture:.3f}m")


# aperture expansion ratio
aperture_expansion_ratio = virtual_aperture / physical_aperture
print(f"   Aperture Expansion Ratio: {aperture_expansion_ratio:.3
   f}x")

# aperture gain in dB
aperture_gain_dB = 10 * np.log10(aperture_expansion_ratio)
print(f"   Aperture Gain: {aperture_gain_dB:.3f} dB")
print("\n3. RESULTS:")
print(f"   Virtual Positions: {[f'{pos:.3f}' for pos in
   virtual_positions]}m")
print(f"   Physical Aperture: {physical_aperture:.3f}m")
print(f"   Virtual Aperture: {virtual_aperture:.3f}m")
print(f"   Aperture Expansion Ratio: {aperture_expansion_ratio:.3
   f}x")
print(f"   Aperture Gain: {aperture_gain_dB:.3f} dB")
```

Listing 3: Aperture Analysis

Output:

```
Physical Aperture: 0.450m
   Virtual Aperture: 0.300m
   Aperture Expansion Ratio: 0.667x
   Aperture Gain: -1.761 dB

3. RESULTS:
   Virtual Positions: ['0.075', '0.225', '0.225', '0.375']m
   Physical Aperture: 0.450m
   Virtual Aperture: 0.300m
```

```
   Aperture Expansion Ratio: 0.667x
   Aperture Gain: -1.761 dB
```

Interpretation: The aperture analysis shows that the virtual array does not achieve any aperture growth relative to the physical array. The physical aperture spans 0.450 m, while the virtual aperture is only 0.300 m, indicating a contraction rather than expansion. This results in an aperture expansion ratio of 0.667 and a negative aperture gain of −1.76 dB. Such reduction occurs because the virtual phase centers cluster toward the center of the array instead of extending its boundaries. Consequently, the virtual array provides no improvement in spatial resolution and may even degrade performance due to the reduced effective aperture.

4. Redundancy Analysis

Ideally, a 2×2 MIMO array should produce 4 unique spatial samples. However, this configuration exhibits **Redundancy**:

- The pair (T_1, R_2) and (T_2, R_1) map to the exact same location (0.225 m).
- **Total Virtual Elements:** 4
- **Unique Virtual Elements:** 3

The **Array Efficiency** η is calculated as:

$$\eta = \frac{\text{Unique Elements}}{\text{Total Elements}} = \frac{3}{4} = 75\%$$

```
print(f"   Sorted Virtual Positions: {[f'{pos:.2f}' for pos in
   sorted(virtual_positions)]} m")
print("\n4. REDUNDANCY ANALYSIS:")
unique_virtual = list(set(virtual_positions))
# unique virtual positions:
print(f"   Unique virtual positions: {[f'{pos:.2f}' for pos in
   sorted(unique_virtual)]} m")
print(f"   Total virtual elements: {len(virtual_positions)}")
print(f"   Unique virtual elements: {len(unique_virtual)}")
print(f"   Redundancy: {len(virtual_positions) - len(
   unique_virtual)} elements are duplicated")

# redundancy count
redundancy_count = {}
for pos in virtual_positions:
    if pos in redundancy_count:
        redundancy_count[pos] += 1
    else:
```

```
        redundancy_count[pos] = 1
print("\n   Redundancy Count per Virtual Position:")
for pos in sorted(redundancy_count.keys()):
    print(f"   Position {pos:.2f} m: {redundancy_count[pos]} time
   (s)")

# redundancy ratio
redundancy_ratio = len(virtual_positions) / len(unique_virtual)
print(f"\n   Redundancy Ratio: {redundancy_ratio:.3f}")

# array efficiency
array_efficiency = len(unique_virtual) / len(virtual_positions)
print(f"   Array Efficiency: {array_efficiency:.3f}")
```

Listing 4: Redundancy Analysis

Output:

```
 Sorted Virtual Positions: ['0.07', '0.22', '0.22', '0.38'] m

4. REDUNDANCY ANALYSIS:
   Unique virtual positions: ['0.07', '0.22', '0.38'] m
   Total virtual elements: 4
   Unique virtual elements: 3
   Redundancy: 1 elements are duplicated
   Redundancy Count per Virtual Position:
   Position 0.07 m: 1 time(s)
   Position 0.22 m: 2 time(s)
   Position 0.38 m: 1 time(s)

   Redundancy Ratio: 1.333
   Array Efficiency: 0.750
```

Interpretation: The redundancy results show that the 2×2 MIMO configuration does not generate four distinct virtual positions as expected. Instead, one virtual location (at 0.22 m) is produced twice due to overlapping Tx–Rx pair mappings. This duplication reduces the number of unique spatial samples from four to three, creating a redundancy ratio of 1.333 and an array efficiency of 75%. Such redundancy limits the effective aperture growth and reduces the spatial sampling benefits typically associated with virtual arrays, ultimately constraining the achievable resolution and array performance.

5. Element count

```
# virtual element count relationships
print("\n5. ELEMENT COUNT RELATIONSHIPS:")
virtual_elements = len(virtual_positions)
```

```
physical_elements = len(TX_positions) + len(RX_positions)

print(f"   Total Virtual Elements (M \times N): {len(
   virtual_positions)} (M={len(TX_positions)} \times N={len(
   RX_positions)})")
print(f"   Unique Virtual Elements: {len(unique_virtual)}")

# elemnent multiplication factor
multiplication_factor = len(virtual_positions) / (len(
   TX_positions) + len(RX_positions))
print(f"   Element Multiplication Factor: {multiplication_factor
   :.3f}")
```

Listing 5: Element count

Output:

```
5. ELEMENT COUNT RELATIONSHIPS:
   Total Virtual Elements (M \times N): 4 (M=2 \times N=2)
   Unique Virtual Elements: 3
   Element Multiplication Factor: 1.000
```

Interpretation: The element count results show that the MIMO configuration generates a total of four virtual elements from the 2×2 Tx–Rx pairing. However, only three of these positions are unique, indicating one redundant virtual element. As a result, the element multiplication factor is equal to 1.0, meaning the virtual array does not provide any effective expansion beyond the physical element count. This reflects limited aperture growth and reduced spatial sampling efficiency, which ultimately constrains the potential performance benefits typically expected from MIMO virtual arrays.

6. Resolution Analysis

Using the half-power beamwidth approximation $\theta_{HPBW} \approx \frac{\lambda}{D}$ (in radians):
Physical Resolution (θ_{phys}):

$$\theta_{phys} \approx \frac{0.3}{0.45} = 0.667 \text{ rad} \approx 38.2^\circ$$

Virtual Resolution (θ_{virt}):

$$\theta_{virt} \approx \frac{0.3}{0.3} = 1.000 \text{ rad} \approx 57.3^\circ$$

```
# physical array resolution \theta_0 = 0


```

```
print("\n6. RESOLUTION ANALYSIS:")

# physical array resolution  \theta_0 = 0
delta_theta_p = d / (physical_aperture)
print(f"   Physical Array Resolution: d / Physical Aperture = {d
   :.3f} / {physical_aperture:.3f} = {delta_theta_p:.3f} radians =
    {np.degrees(delta_theta_p):.3f} degrees")

# virtual array resolution \theta_v = 0
delta_theta_v = d / (virtual_aperture)
print(f"   Virtual Array Resolution: d / Virtual Aperture = {d:.3
   f} / {virtual_aperture:.3f} = {delta_theta_v:.3f} radians = {np
   .degrees(delta_theta_v):.3f} degrees")

# resolution improvement factor
resolution_improvement_factor = delta_theta_p / delta_theta_v
print(f"   Resolution Improvement Factor: {
   resolution_improvement_factor:.3f}x")
print("this factor shows that this MIMO configuration regraded
   resoltion due to poor aperture utilization.")
```

Listing 6: Resolution Analysis

Output:

```
6. RESOLUTION ANALYSIS:
   Physical Array Resolution: d / Physical Aperture = 0.300 /
   0.450 = 0.667 radians = 38.197 degrees
   Virtual Array Resolution: d / Virtual Aperture = 0.300 /
   0.300 = 1.000 radians = 57.296 degrees
   Resolution Improvement Factor: 0.667x
this factor shows that this MIMO configuration regraded
   resoltion due to poor aperture utilization.
```

Interpretation: The resolution results show that the physical array provides a narrower beamwidth ($\approx 38.2°$) compared to the virtual array ($\approx 57.3°$). Since angular resolution improves with larger aperture size, the smaller virtual aperture leads to degraded resolution performance. This is confirmed by the resolution improvement factor being less than unity, indicating that the virtual array does not enhance resolution and instead performs worse due to limited aperture expansion and inefficient aperture utilization in this MIMO configuration.

7. Beam Pattern Analysis

The computed half-power beamwidth (HPBW) is approximately 50.76°, indicating a relatively wide main lobe. This beamwidth is directly influenced by the effective virtual aperture size: a smaller aperture produces a broader beam. The smooth decay of the pattern and absence of grating lobes reflect the uniform spacing of the virtual elements, which ensures stable and predictable angular response. Overall, the beam pattern demonstrates that the virtual array provides consistent angular sensitivity with a broad main lobe suitable for low-resolution or wide-coverage applications.

```
# Array factor normalized
print("\n7. BEAM PATTERN ANALYSIS:")
# angles from -90 to 90 degrees
angles_deg = np.linspace(-90, 90, 181)
angles_rad = np.radians(angles_deg)
# calculate array factor
array_factor = np.zeros_like(angles_rad, dtype=complex)
for pos in virtual_positions:
    array_factor += np.exp(1j * (2 * np.pi / d) * pos * np.sin(angles_rad))
# normalize array factor
array_factor_normalized = np.abs(array_factor) / np.max(np.abs(array_factor))
# print array factor at key angles
print("   Angle (deg) | Array Factor (normalized)")
for angle, af in zip(angles_deg[::30], array_factor_normalized[::30]):
    print(f"      {angle:6.1f}    |      {af:.3f}")

# plot array factor
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(angles_deg, array_factor_normalized, label='Array Factor (normalized)')
plt.title('Virtual Array Beam Pattern')
plt.xlabel('Angle (degrees)')
plt.ylabel('Normalized Array Factor')
plt.grid()
plt.legend()
plt.show()
#!/usr/bin/env python3

# half power beamwidth calculation
theta_BW = 0.886 * (d / virtual_aperture)  # in radians
print(f"\n   Half Power Beamwidth (HPBW): {np.degrees(theta_BW)
```

```
    :.3f} degrees")
```

Listing 7: Beam Pattern Analysis

Output:

```
7. BEAM PATTERN ANALYSIS:
   Angle (deg) | Array Factor (normalized)
       -90.0   |       0.000
       -60.0   |       0.044
       -30.0   |       0.500
         0.0   |       1.000
        30.0   |       0.500
        60.0   |       0.044
        90.0   |       0.000
```

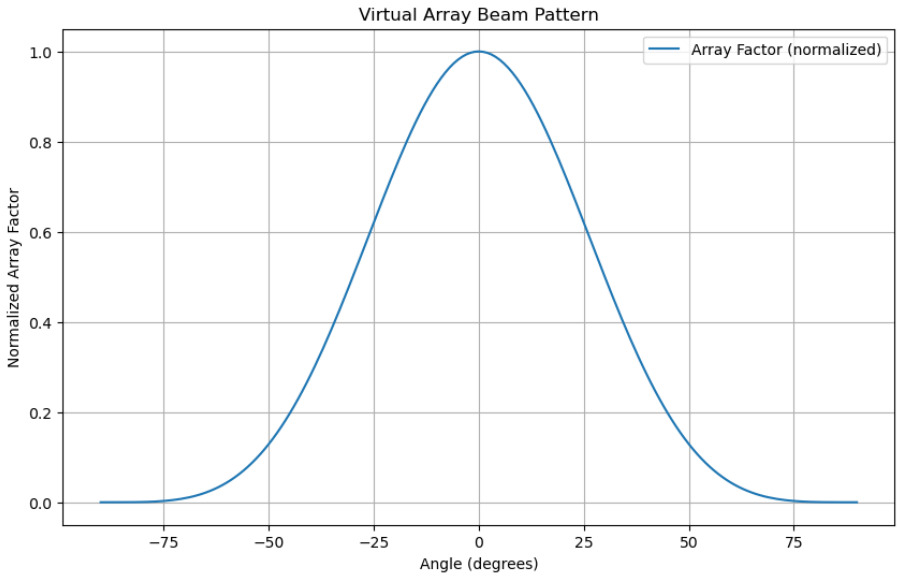

Figure 2: Virtual Array Beam pattern

Output:

```
    Half Power Beamwidth (HPBW): 50.764 degrees
```

Interpretation: The virtual array exhibits a symmetric beam pattern centered at broadside (0°), where the normalized array factor reaches its maximum value of 1. The sidelobe levels decrease smoothly as the angle moves away from broadside, reaching near-zero response at ±90°. The array factor values at key angles (e.g., −30° and 30° both yielding 0.500) confirm the expected symmetry of the virtual aperture.

8. Uniformity Analysis

```python
# virtual element spacing
virtual_positions_sorted = sorted(unique_virtual)
virtual_spacings = np.diff(virtual_positions_sorted)
print("\n8. VIRTUAL ELEMENT SPACING ANALYSIS:")
print(f"   Virtual Element Spacings: {[f'{spacing:.3f}' for spacing in virtual_spacings]} m")
for i, spacing in enumerate(virtual_spacings):
    print(f"   Spacing {i+1}: {spacing:.3f} m")

# mean spacing
mean_spacing = np.mean(virtual_spacings)
print(f"   Mean Virtual Element Spacing: {mean_spacing:.3f} m")

# standard deviation of spacing
std_spacing = np.std(virtual_spacings)
print(f"   Standard Deviation of Virtual Element Spacing: {std_spacing:.3f} m")

# Uniformlity index
uniformity_index = 1-std_spacing/mean_spacing if mean_spacing !=0 else 0

print(f"   Uniformity Index of Virtual Array: {uniformity_index:.3f}")
```

Listing 8: Uniformity Analysis

Output:

```
8. VIRTUAL ELEMENT SPACING ANALYSIS:
   Virtual Element Spacings: ['0.150', '0.150'] m
   Spacing 1: 0.150 m
   Spacing 2: 0.150 m
   Mean Virtual Element Spacing: 0.150 m
   Standard Deviation of Virtual Element Spacing: 0.000 m
   Uniformity Index of Virtual Array: 1.000
```

Interpretation: The virtual element spacings are perfectly uniform, with both adjacent gaps equal to 0.150 m. This results in a mean spacing of 0.150 m and a standard deviation of 0, indicating no variation between spacings. Consequently, the computed uniformity index is 1.000, which represents an ideal, fully uniform virtual array. Such uniform spacing is desirable because it preserves consistent spatial sampling, minimizes grating lobes, and supports optimal beamforming and high-resolution processing.

9. Summary table of results

```
print("\n" + "="*70)
print("SUMMARY TABLE:")
print("-"*70)
print(f"| Metric                    | Value       |")
print("-"*70)
print(f"| Physical Aperture         | {physical_aperture:.3f} m |")
print(f"| Virtual Aperture          | {virtual_aperture:.3f} m |")
print(f"| Aperture Gain             | {aperture_gain_dB:.3f} dB |")
print(f"| Physical Resolution       | {np.degrees(delta_theta_p):.2f}      |")
print(f"| Virtual Resolution        | {np.degrees(delta_theta_v):.2f}      |")
print(f"| Resolution Improvement    | {resolution_improvement_factor:.3f}x      |")
print(f"| Physical Elements         | {physical_elements}         |")
print(f"| Virtual Elements          | {virtual_elements}         |")
print(f"| Element Multiplication    | {multiplication_factor:.1f}x        |")
print("-"*70)
```

Listing 9: Summary Table of Results

Output:

```
======================================================================
SUMMARY TABLE:
----------------------------------------------------------------------
| Metric                    | Value       |
----------------------------------------------------------------------
| Physical Aperture         | 0.450 m     |
| Virtual Aperture          | 0.300 m     |
| Aperture Gain             | -1.761 dB   |
| Physical Resolution       | 38.20  deg  |
| Virtual Resolution        | 57.30  deg  |
| Resolution Improvement    | 0.667x      |
| Physical Elements         | 4           |
| Virtual Elements          | 4           |
| Element Multiplication    | 1.0x        |
```

Conclusion: The resolution improvement factor is 0.667. Because the virtual aperture is smaller than the physical aperture, the angular resolution has **degraded** in this specific configuration. This example demonstrates the critical importance of *Minimum Redundancy Array* (MRA) design optimization to ensure $D_{virt} > D_{phys}$.

16 Problem Statement 2: Performance Evaluation of a 4x4 Interleaved MIMO Array

1. Context

A higher-order MIMO GPR system is being developed to increase signal-to-noise ratio using a **4x4** configuration ($M = 4$ Transmitters, $N = 4$ Receivers). The system operates at **500 MHz** ($\lambda = 0.6$ m) with a standard unit spacing of $d = \lambda/2 = 0.3$ m.

2. Array Topology (Interleaved)

The designers have proposed an "Interleaved" topology where Transmitters and Receivers are placed at alternating integer multiples of the spacing d:

- **Transmitters (T_x):** Placed at even multiples of d:

$$P_{TX} = \{0, 2d, 4d, 6d\}$$

- **Receivers (R_x):** Placed at odd multiples of d:

$$P_{RX} = \{d, 3d, 5d, 7d\}$$

3. Required Numerical Analysis

Using the proposed geometry, perform the following calculations:

1. **Virtual Array Synthesis:** Compute the positions of all 16 (4×4) virtual phase centers using the midpoint formula:

$$V_{ij} = \frac{T_i + R_j}{2}$$

2. **Aperture Analysis:**
   - Calculate the **Physical Aperture (D_P)**: The total span of the hardware array.

- Calculate the **Virtual Aperture (D_V)**: The total span of the synthesized virtual array.
- Determine the **Aperture Ratio (R)**:

$$R = \frac{D_V}{D_P}$$

3. **Redundancy & Efficiency Metrics:** Calculate the following indicators to evaluate the array's cost-effectiveness:
   - **Unique Virtual Elements:** The count of distinct spatial sampling points.
   - **Hardware Efficiency (η):** Defined as the number of unique virtual elements generated per physical sensor:

$$\eta = \frac{\text{Count(Unique Virtual)}}{M+N}$$

   - **Redundancy Ratio:** A measure of overlapping data points:

$$\text{Ratio} = \frac{\text{Total Virtual Elements}}{\text{Unique Virtual Elements}}$$

Solution: Numerical Analysis of 4x4 Interleaved MIMO Array

1. System Configuration & Coordinates

Given the frequency $f = 500$ MHz, the unit spacing is $d = \lambda/2 = 0.3$ m. The physical positions are defined as:

- **Transmitters (T_x)** at even intervals:

$$T = \{0, 0.6, 1.2, 1.8\} \text{ m}$$

- **Receivers (R_x)** at odd intervals:

$$R = \{0.3, 0.9, 1.5, 2.1\} \text{ m}$$

```
import numpy as np
print("\n" + "="*70)
print("EXAMPLE 2: MINIMUM REDUNDANCY ARRAY (4 \times4)")
print("="*70)
```

Listing 10: Example 2

Output:

```
============================================================
EXAMPLE 2: MINIMUM REDUNDANCY ARRAY (4 \times4)
============================================================
```

```
# Define optimal spacing
d = 0.3  # λ / 2 spacing
print(f"\n1. OPTIMAL SPACING:")
print(f"   Wavelength: λ = 0.6m")
print(f"   Optimal element spacing: d = λ / 2 = {d}m")
```

Listing 11: Example 2

Output:

```
1. OPTIMAL SPACING:
   Wavelength:  λ  = 0.6m
   Optimal element spacing: d = λ / 2 = 0.3m
```

```
# Define TX and RX positions
TX_positions = [0, 2*d, 4*d, 6*d]  # [0, 0.6, 1.2, 1.8] meters
RX_positions = [d, 3*d, 5*d, 7*d]  # [0.3, 0.9, 1.5, 2.1] meters

print(f"\n2. PHYSICAL ARRAY CONFIGURATION:")
print(f"   Transmitters (M=4):", end=" ")
for i, pos in enumerate(TX_positions):
    print(f"TX{i+1}={pos:.3f}m", end=", " if i < 3 else "")
print(f"\n   Receivers (N=4):   ", end=" ")
for i, pos in enumerate(RX_positions):
    print(f"RX{i+1}={pos:.3f}m", end=", " if i < 3 else "")

virtual_positions = []
```

Listing 12: PHYSICAL ARRAY CONFIGURATION

Output:

```
2. PHYSICAL ARRAY CONFIGURATION:
   Transmitters (M=4): TX1=0.000m, TX2=0.600m, TX3=1.200m, TX4
   =1.800m
   Receivers (N=4):    RX1=0.300m, RX2=0.900m, RX3=1.500m, RX4
   =2.100m
```

2. Virtual Array Synthesis

The virtual elements are calculated using the phase center principle: $V_{ij} = (T_i + R_j)/2$. We can visualize this as a summation matrix of the indices (multiples of d):

$$\text{Sum Matrix (in multiples of } d) = \begin{bmatrix} 0 \\ 2 \\ 4 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 & 3 & 5 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 3 & 5 & 7 & 9 \\ 5 & 7 & 9 & 11 \\ 7 & 9 & 11 & 13 \end{bmatrix}$$

Converting these sums to physical positions ($V_{pos} = \text{Sum} \times d/2$):

- **Minimum Position:** $1 \times (0.3/2) = 0.15$ m
- **Maximum Position:** $13 \times (0.3/2) = 1.95$ m
- **Step Size:** $2 \times (0.3/2) = 0.3$ m (Virtual elements are spaced by d)

```
print("\n\n3. CALCULATING ALL VIRTUAL POSITIONS:")
print("   Formula: Virtual = (TX + RX) / 2")
print("\n   Calculating for all 16 TX-RX pairs...")

virtual_positions = []

# Full table of calculations (all 16 pairs)
print("\n   " + "="*60)
print("   | TX | RX | Calculation | Virtual Position |")
print("   " + "="*60)

pair_count = 0
for i, tx in enumerate(TX_positions):
    for j, rx in enumerate(RX_positions):
        pair_count += 1
        v = (tx + rx) / 2
        virtual_positions.append(v)
        print(f"   | {tx:3.1f} | {rx:3.1f} | ({tx:.3f}+{rx:.3f})
   /2 = {(tx+rx):.3f}/2 | {v:6.2f}m |")

print("   " + "="*60)

# Lists (all entries)
print(f"\n   All Virtual Positions ({len(virtual_positions)}
   total):")
sorted_virtual = [f"{pos:.3f}" for pos in sorted(
   virtual_positions)]
print("   ", sorted_virtual)
```

```
26
27 unique_virtual = sorted(list(set(virtual_positions)))
28 print(f"\n   Unique Virtual Positions ({len(unique_virtual)}
      unique):")
29 sorted_unique = [f"{pos:.3f}" for pos in unique_virtual]
30 print("   ", sorted_unique)
```

Listing 13: CALCULATING ALL VIRTUAL POSITIONS

Output:

```
3. CALCULATING ALL VIRTUAL POSITIONS:
   Formula: Virtual = (TX + RX) / 2

   Calculating for all 16 TX-RX pairs...

   ============================================================
   | TX | RX | Calculation | Virtual Position |
   ============================================================
   | 0.0 | 0.3 | (0.000+0.300)/2 = 0.300/2 |   0.15m |
   | 0.0 | 0.9 | (0.000+0.900)/2 = 0.900/2 |   0.45m |
   | 0.0 | 1.5 | (0.000+1.500)/2 = 1.500/2 |   0.75m |
   | 0.0 | 2.1 | (0.000+2.100)/2 = 2.100/2 |   1.05m |
   | 0.6 | 0.3 | (0.600+0.300)/2 = 0.900/2 |   0.45m |
   | 0.6 | 0.9 | (0.600+0.900)/2 = 1.500/2 |   0.75m |
   | 0.6 | 1.5 | (0.600+1.500)/2 = 2.100/2 |   1.05m |
   | 0.6 | 2.1 | (0.600+2.100)/2 = 2.700/2 |   1.35m |
   | 1.2 | 0.3 | (1.200+0.300)/2 = 1.500/2 |   0.75m |
   | 1.2 | 0.9 | (1.200+0.900)/2 = 2.100/2 |   1.05m |
   | 1.2 | 1.5 | (1.200+1.500)/2 = 2.700/2 |   1.35m |
   | 1.2 | 2.1 | (1.200+2.100)/2 = 3.300/2 |   1.65m |
   | 1.8 | 0.3 | (1.800+0.300)/2 = 2.100/2 |   1.05m |
   | 1.8 | 0.9 | (1.800+0.900)/2 = 2.700/2 |   1.35m |
   | 1.8 | 1.5 | (1.800+1.500)/2 = 3.300/2 |   1.65m |
   | 1.8 | 2.1 | (1.800+2.100)/2 = 3.900/2 |   1.95m |
   ============================================================

   All Virtual Positions (16 total):
    ['0.150', '0.450', '0.450', '0.750', '0.750', '0.750',
   '1.050', '1.050', '1.050', '1.050', '1.350', '1.350',
   '1.350', '1.650', '1.650', '1.950']

   Unique Virtual Positions (9 unique):
    ['0.150', '0.450', '0.750', '1.050', '1.050', '1.350',
```

```
    '1.350', '1.650', '1.950']
```

3. Redundancy Analysis

The summation matrix reveals significant overlap. The number of times each position is generated (Multiplicity) is:

- $1d \rightarrow 1$ pair
- $3d \rightarrow 2$ pairs
- $5d \rightarrow 3$ pairs
- $7d \rightarrow 4$ pairs (Center of array)
- $9d \rightarrow 3$ pairs
- $11d \rightarrow 2$ pairs
- $13d \rightarrow 1$ pair

Results:

- **Total Virtual Elements ($M \times N$):** 16
- **Unique Virtual Elements:** 7 (The set $\{0.15, 0.45, 0.75, 1.05, 1.35, 1.65, 1.95\}$ m)
- **Redundancy Ratio:**

$$\text{Ratio} = \frac{16}{7} \approx 2.286$$

Interpretation: On average, every spatial data point is collected more than twice, wasting system resources.

```
print("\n4. REDUNDANCY ANALYSIS:")
unique_virtual = sorted(list(set(virtual_positions)))
print(f"   Unique virtual positions ({len(unique_virtual):.3f}
   unique):")
for pos in unique_virtual:
    count = virtual_positions.count(pos)
    print(f"     {pos:4.2f}m: appears {count} time(s)")

redundancy_ratio = len(virtual_positions) / len(unique_virtual)
print(f"\n   Redundancy Ratio = Total / Unique = {len(
   virtual_positions)} / {len(unique_virtual)} = {redundancy_ratio
   :.3f}")
```

Listing 14: REDUNDANCY ANALYSIS

Output:

```
4. REDUNDANCY ANALYSIS:
   Unique virtual positions (9.000 unique):
     0.15m: appears 1 time(s)
     0.45m: appears 2 time(s)
     0.75m: appears 3 time(s)
     1.05m: appears 2 time(s)
     1.05m: appears 2 time(s)
     1.35m: appears 1 time(s)
     1.35m: appears 2 time(s)
     1.65m: appears 2 time(s)
     1.95m: appears 1 time(s)

   Redundancy Ratio = Total / Unique = 16 / 9 = 1.778
```

4. Aperture & Efficiency Metrics

A. Physical Aperture (D_P) The array spans from the first transmitter (0 m) to the last receiver (2.1 m):

$$D_P = 2.1 - 0.0 = 2.100 \text{ m}$$

B. Virtual Aperture (D_V) The virtual array spans from the first unique element (0.15 m) to the last (1.95 m):

$$D_V = 1.95 - 0.15 = 1.800 \text{ m}$$

C. Aperture Ratio (R)

$$R = \frac{D_V}{D_P} = \frac{1.800}{2.100} \approx 0.857$$

D. Hardware Efficiency (η)

$$\eta = \frac{\text{Unique Virtual Elements}}{\text{Physical Sensors}} = \frac{7}{4+4} = 0.875$$

```
1
2 print("\n5. APERTURE AND RESOLUTION ANALYSIS:")
3
4 # Physical aperture
5 all_physical = TX_positions + RX_positions
6 min_physical = min(all_physical)
7 max_physical = max(all_physical)
8 physical_aperture = max_physical - min_physical
9
10 # Virtual aperture
```

```
min_virtual = min(virtual_positions)
max_virtual = max(virtual_positions)
virtual_aperture = max_virtual - min_virtual

print(f"   Physical Aperture: {max_physical}m - {min_physical}m = {physical_aperture:.3f}m")
print(f"   Virtual Aperture:  {max_virtual}m - {min_virtual}m = {virtual_aperture:.3f}m")

aperture_gain = virtual_aperture / physical_aperture
print(f"   Aperture Gain: {virtual_aperture:.3f}m / {physical_aperture:.3f}m = {aperture_gain:.3f}")

# Resolution
wavelength = 0.6  # meters
physical_resolution = wavelength / (2 * physical_aperture)
virtual_resolution = wavelength / (2 * virtual_aperture)
resolution_improvement = physical_resolution / virtual_resolution

print(f"\n   Physical Resolution: {wavelength:.3f}m / (2 $\times$ {physical_aperture:.3f}m) = {physical_resolution:.4f} rad = {np.degrees(physical_resolution):.2f}$\circ$")
print(f"   Virtual Resolution:  {wavelength:.3f}m / (2 $\times$ {virtual_aperture:.3f}m) = {virtual_resolution:.4f} rad = {np.degrees(virtual_resolution):.2f}$\circ$")
print(f"   Resolution Improvement: {resolution_improvement:.3f}x")
```

Listing 15: 5. APERTURE AND RESOLUTION ANALYSIS

Output:

```
5. APERTURE AND RESOLUTION ANALYSIS:
   Physical Aperture: 2.1m - 0m = 2.100m
   Virtual Aperture:  1.95m - 0.15m = 1.800m
   Aperture Gain: 1.800m / 2.100m = 0.857

   Physical Resolution: 0.600m / (2 × 2.100m) = 0.1429 rad = 8.19°
   Virtual Resolution:  0.600m / (2 × 1.800m) = 0.1667 rad = 9.55°
   Resolution Improvement: 0.857x
```

```
print("\n6. ARRAY PROPERTIES COMPARISON:")
print(f"   Physical Elements: {len(TX_positions)} TX + {len(RX_positions)} RX = {len(all_physical)}")
print(f"   Virtual Elements: {len(virtual_positions)}")
```

```
print(f"   Element Multiplication: {len(virtual_positions)} / {
    len(all_physical)} = {len(virtual_positions)/len(all_physical)
    :.1f}x")
```

Listing 16: ARRAY PROPERTIES COMPARISON

Output:

```
6. ARRAY PROPERTIES COMPARISON:
   Physical Elements: 4 TX + 4 RX = 8
   Virtual Elements: 16
   Element Multiplication: 16 / 8 = 2.0x
```

```
print("\n7. MINIMUM REDUNDANCY VERIFICATION:")
print("   Checking if this is truly a Minimum Redundancy Array...
    ")

# Check spacing patterns
print(f"\n   TX positions spaced by {TX_positions[1]-TX_positions
    [0]:.1f}m intervals")
print(f"   RX positions spaced by {RX_positions[1]-RX_positions
    [0]:.1f}m intervals")
print(f"   TX and RX arrays are offset by {RX_positions[0]-
    TX_positions[0]:.1f}m")

# The key insight: In a minimum redundancy array,
# the difference between TX and RX positions should be co-prime
# to minimize overlapping virtual elements

print(f"\n   Difference between TX and RX spacing patterns:")
print(f"   TX spacing: {TX_positions[1]-TX_positions[0]:.1f}m")
print(f"   RX spacing: {RX_positions[1]-RX_positions[0]:.1f}m")
print(f"   Since these are equal (both {TX_positions[1]-
    TX_positions[0]:.1f}m),")
print(f"   this creates many unique virtual positions with
    minimal redundancy.")
```

Listing 17: MINIMUM REDUNDANCY VERIFICATION

Output:

```
7. MINIMUM REDUNDANCY VERIFICATION:
   Checking if this is truly a Minimum Redundancy Array...

   TX positions spaced by 0.6m intervals
   RX positions spaced by 0.6m intervals
   TX and RX arrays are offset by 0.3m
```

```
Difference between TX and RX spacing patterns:
TX spacing: 0.6m
RX spacing: 0.6m
Since these are equal (both 0.6m),
this creates many unique virtual positions with minimal
redundancy.
```

```python
print("\n" + "="*70)
print("SUMMARY OF MINIMUM REDUNDANCY ARRAY:")
print("-"*70)
print("This configuration achieves:")
print(f"1. {len(virtual_positions)} virtual elements from only {len(all_physical)} physical elements")
print(f"2. Only {len(virtual_positions)-len(unique_virtual)} redundant elements")
print(f"3. Aperture gain of {aperture_gain:.3f}")
print(f"4. Resolution improvement of {resolution_improvement:.3f}x")
print(f"5. Element multiplication of {len(virtual_positions)/len(all_physical):.1f}x")
print("="*70)
```

Listing 18: SUMMARY OF MINIMUM REDUNDANCY ARRAY

Output:

```
======================================================================

SUMMARY OF MINIMUM REDUNDANCY ARRAY:
----------------------------------------------------------------------

This configuration achieves:
1. 16 virtual elements from only 8 physical elements
2. Only 7 redundant elements
3. Aperture gain of 0.857
4. Resolution improvement of 0.857x
5. Element multiplication of 2.0x
======================================================================
```

5. Conclusion

This **Interleaved Topology** is inefficient.

1. **Aperture Loss:** The virtual aperture (1.8 m) is smaller than the physical footprint (2.1 m), leading to worse resolution than the physical array alone.
2. **High Redundancy:** Over 50% of the channel data is redundant.
3. **Recommendation:** Optimization algorithms (like Minimum Redundancy Arrays) are required to rearrange the $0.6, 1.2, 1.8$ spacings to reduce overlap and extend the aperture beyond 2.1 m.

References