DYNAMIC BEAMFORMING WITHOUT PHASE SHIFTERS

A Thesis

by

TAAHIR DAVID AHMED

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Dylan Shell
Co-Chair of Committee,	Gregory Huff
Committee Members,	Radu Stoleru
	Dezhen Song
	Jean-Francois Chamberland-Tremblay
Head of Department,	Dilma Da Silva

May 2017

Major Subject: Computer Engineering

ABSTRACT

Array beamforming is a widely-used technique for producing a flexible, retargetable, and tunable beam using fixed-position transmitting or receiving elements. It is well-understood, and widely used. Distributed and collaborative versions of the problem — where elements do not have fixed locations known ahead of time, elements may move, or elements may need to cooperate without central control — offer the benefits of beamforming in a much wider set of scenarios. The greater complexity introduced by the required coordination and estimation have limited its use in deployed systems.

One of the common assumptions of existing distributed beamforming techniques is that each element can use a phase shifter to alter its intrinsic transmit phase to an optimal value (measured, for closed loop strategies; calculated, for open-loop strategies) that minimizes inter-element interference in the main beam. In this thesis, I present a simple alternative method for minimizing interference: elements should deactivate their transmitter if their intrinsic phase is too distant from their optimal phase.

Omitting the phase shifter permits cheaper elements, but requires overprovisioning the number of elements in the beamforming system. In systems that have many elements *a priori* (a multirobot swarm, for example), this deactivation strategy can add beamforming capability at a lower cost.

I analyze the general behavior of a beamforming system built using this principle, discuss the tradeoff between allowable phase error and power in the main beam, and prove a lower bound on fraction of active elements in a transmitter swarm. In addition, I present simulation and benchtop results that illustrate the feasibility of

element deactivation as a beamforming technique.

DEDICATION

To my family.

ACKNOWLEDGMENTS

Special thanks are due to Dr. Dylan Shell and Dr. Gregory Huff, who have been patient and kind throughout the long and slow process of my thesis work.

Additionally, thanks go to my fellow students in Drs. Shell and Huff's research groups. I can only hope that I have given as much help as I received.

Finally, thanks are due to my parents for encouraging me, and especially to my wife Katherine for her relentless support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Dylan Shell (advisor) of the Department of Computer Science and Engineering, Professor Gregory Huff (co-advisor) of the Department of Electrical and Computer Engineering, Professors Dezhen Song and Radu Stoleru of the Department of Computer Science and Engineering, and Professor Jean-Francois Chamberland-Tremblay of the Department of Electrical and Computer Engineering.

All work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a Research Assistantship from Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

The array beamforming problem is the process of forming a strong beam in a given direction using a collection of antennas that do not necessarily have any preferential direction. By controlling the amplitude and phase of each antenna, their individual radiation patterns can be made to constructively interfere in a chosen direction, forming a main beam. At the same time, their radiation patterns interfere destructively in other directions, ensuring that most of the emitted power travels along the main beam.

The benefits of array beamforming are broadly applicable. For example:

- A team of autonomous rovers needs to communicate with an overhead satellite without removing rovers from their current tasks. Dynamic beamforming would allow them to assemble a high-gain beam capable of communicating with the satellite using only the low-gain antennas used for inter-rover communication.

- A wireless sensor network needs to report to a remote base station, beyond the range of any individual module. Dynamic beamforming would allow the sensor modules to pool their transmit power, even though they may be widely separated.

Despite the promised benefits, random and dynamic beamforming techniques are not widely used in deployed systems — most strategies require intensive communication between elements, as well as fairly sophisticated hardware.

Basic approaches to the beamforming problem assume control over element positions, transmit amplitudes, and transmit phases [16]. Random arrays [10] sacrifice control over element positions, while maintaining the main beam, and dynamic ar-

rays support actively steering and retargeting the beam. The underlying math of all these variants is largely the same; only the implementation technique changes.

To produce a coherent beam, the transmitting elements must all be in phase as a wave front travels along the beam axis (See Chapter 2 for an explanation). Existing approaches (*classical steering*) accomplish this goal by giving each element a phase shifter to correct the difference between the element's intrinsic phase and the phase required by its position in the array.

I propose *deactivation steering* as a simple alternative: if the correction that an element's phase shifter would have applied is greater than a threshold, then the element deactivates its transmitter. Deactivation steering removes the requirement for phase shifters at the cost of reduced beam power for a given number of elements.

Removing the per-element phase shifter will lower the cost and complexity of the resulting system. Hopefully, this will remove one of the barriers that stands in the way of wider deployment of systems based on dynamic beamforming techniques.

The remainder of this thesis:

- Formalizes the dynamic beamforming problem, classical steering, and deactivation steering (Chapter 2);
- Covers specific existing approaches to dynamic beamforming (Chapter 3);
- Discusses the expected performance of a beam under deactivation steering, compares it to that of classical steering, and proves a lower bound on the number of active elements (Chapter 4);
- Presents the results of benchtop experiments that show the basic feasibility and realistic shortcomings of the deactivation strategy (Chapter 5).

2. FORMALIZATION OF DEACTIVATION STEERING

In order for beamforming to occur, wavefronts from each element must arrive at the same point on the main beam axis close together in time. Figure 2.1 illustrates this process for a classical periodic array. The standard tool for ensuring this occurs for any given array is a quantity called the *array factor* [16]:

$$\mathrm{AF}(\hat{q}) = \sum_i s_i \mathrm{e}^{-j(k\hat{q}\cdot\vec{p_i}+\phi_i)}, \tag{2.1}$$

where

- $\hat{q}$ is the query direction (a unit vector);
- s_i is the *steering coefficient* of element i, a complex number embodying the input amplitude and phase of element i;
- k is the wave number, $\frac{2\pi}{\lambda}$;
- $\vec{p_i}$ is the position vector of element i;

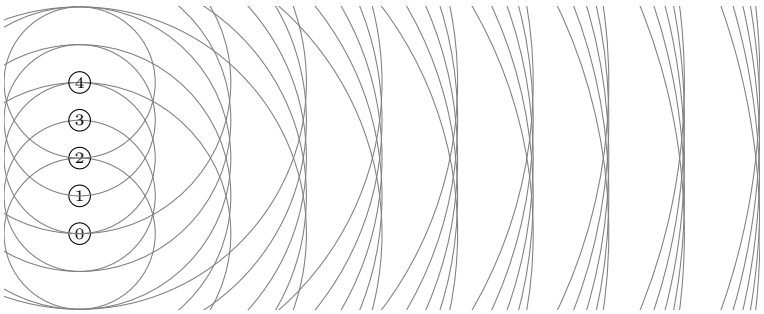


Figure 2.1: Constructive interference in a classical antenna array. As you move further away from the array along the beam axis, the wavefronts from each element arrive closer and closer together.

- ϕ_{0i} is the intrinsic phase offset of element i.

The array factor characterizes the radiation pattern of an array in a way that is independent of the radiation patterns of the individual elements, making it useful for comparing different types of arrays.

The set of coefficients s_i, the *steering vector*, describes the input signal (amplitude and phase) of element i. If the element positions are not controllable, then the steering vector is the only controllable term in the array factor.

The complex angles involved in the array factor can be visualized in *phase space*, as angular positions on the unit circle. In the case where the array is unsteered (all s_i are real), the phase space position of element i is:

$$\phi_i = (k\hat{q} \cdot \vec{p_i}) + \phi_{0i} \mod 2\pi. \tag{2.2}$$

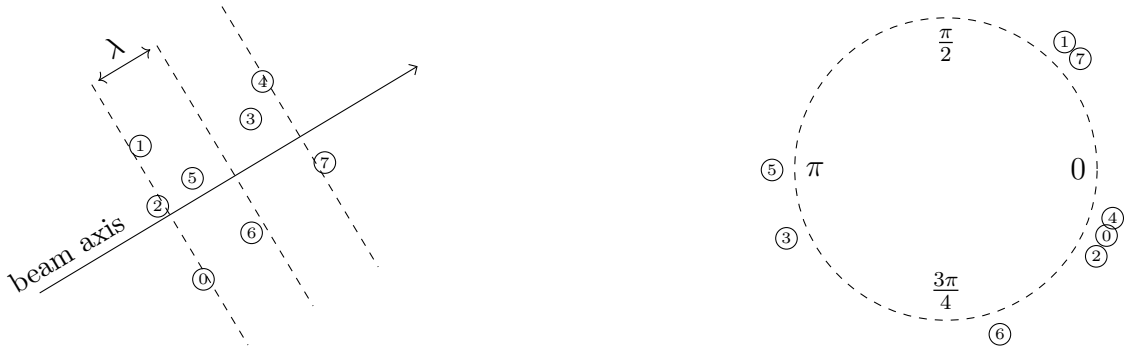
The quantity $(k\hat{q} \cdot \vec{p_i})$ can be called the *distance phase offset*, since it measures the distance (relative to the origin) of element i along the query axis, modulo the wavelength. Note that the phase space position of element i is dependent on the query vector $\hat{q}$.

Figure 2.2 illustrates the correspondence between $\vec{p_i}$ and ϕ_i. Elements that are diametrically opposed on the phase-space diagram interfere destructively, while those close together interfere constructively.

The typical way to form a beam from a given set of element positions, here called *classical steering*, sets the steering vector to be:

$$s_{c,i} = A\mathrm{e}^{j(k\hat{b}\cdot\vec{p_i}+\phi_{0i})} \tag{2.3}$$

where $\hat{b}$ is the beam direction (a unit vector), and A is a fixed amplitude, identical for



(a) 2D Beamforming (b) Phase space

Figure 2.2: (a) Beamforming with eight randomly-placed transmitters. Assuming that all elements have a common clock and no intrinsic phase offset ($\phi_{0i} = 0$), (b) shows the corresponding phase-space positions.

all elements [10, 6]. As with the array factor, the steering vector is usually described in spherical coordinates, without consideration of an element's intrinsic phase offset. The complex magnitude of $s_{\mathrm{c},i}$ is fixed, indicating that all elements transmit with the same amplitude. The complex phase of $s_{\mathrm{c},i}$ counteracts each element's distance and intrinsic phase offsets, effectively forcing all elements to $\phi_i = 0$ in phase space. Because of this, the elements then interfere constructively along the main beam.

The fact that $s_{\mathrm{c},i}$ has an arbitrary complex angle implies that any physical implementation of classical steering requires a phase shifter for each antenna element.

To work around this requirement, I propose *deactivation steering*:

$$s_{\mathrm{d},i} = \begin{cases} 1, & \text{if } k\hat{b} \cdot \vec{p_i} + \phi_{0i} \in [\phi_{\max}, \phi_{\max} + w) \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

The half-open interval $[\phi_{\max}, \phi_{\max} + w)$, called the *maximal interval*, is the section of phase space (of width w) that contains the most elements (Figure 2.3). In general, there is more than one maximal interval of width w for any given set of element

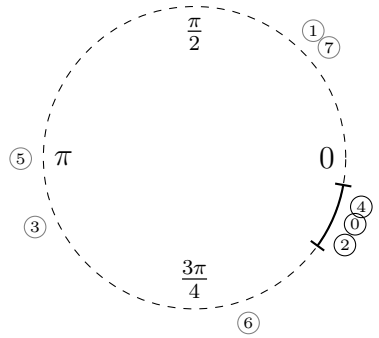


Figure 2.3: An example maximal interval ($w = 10\% \cdot 2\pi$) as Figure 2.2. The interval covers the maximum number of transmitters possible.

positions.

The maximal interval width w is the maximum phase error the system is willing to tolerate. It is a freely-chosen design parameter. Figure 2.4 shows the result of applying deactivation steering to a set of elements.

The primary advantage of deactivation steering is that all coefficients $s_{\mathrm{d},i}$ are purely real, indicating that no phase shifters are required. In effect, deactivation steering achieves coherence by turning off all elements that are poorly placed for contributing to a given beam.

Deactivation steering is founded on several assumptions:

1. There are many more transmitters available than are needed to form a sufficiently powerful beam.

2. Mutual coupling between receivers, multipath fading, and shadowing are all assumed not to occur.

3. Each transmitter has access to an estimate of its phase space position. This can be derived from a real position estimate and Equation 2.2, but there are also strategies that allow a direct estimate of phase space position.

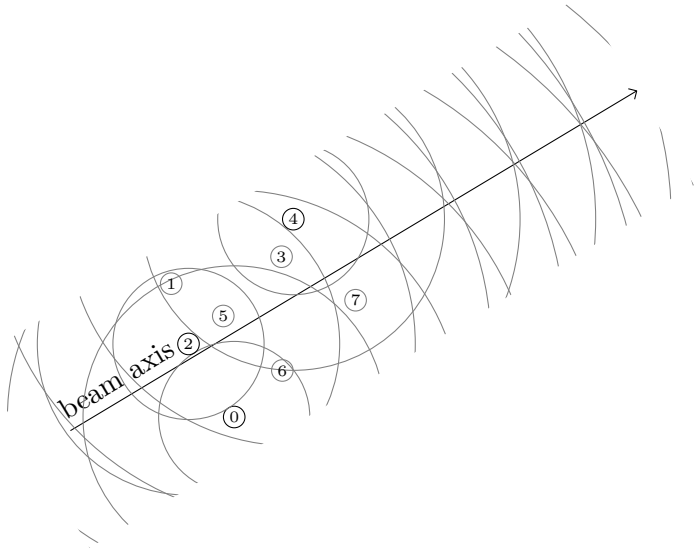


Figure 2.4: The result of activating the transmitters from Figure 2.3. Observe that the wavefronts become more coincident the further along the beam axis they travel.

4. All transmitters have access to a common local oscillator (clock) signal.

Assumption 1 is unique to deactivation steering, while assumptions 2, 3, and 4 are shared, to a large degree, among all beamforming strategies in the literature.

3. EXISTING WORK

The random array problem has been studied since the early 1960s, when ever-more-complicated periodic array spacing strategies were being pursued in an attempt to optimize array performance. Y. T. Lo showed [10] that positioning elements randomly across the array aperture requires far fewer transmitting elements to maintain the same level of average array performance. Since then, random arrays have been the subject of incredibly deep research.

Methods for automatically synchronizing a phased array are nearly as old as phased arrays themselves. In the general case, these *adaptive* methods work by exposing the array to the signal from a transmitter in the far field. The phase of the signal can be measured at each array antenna, and the reciprocity theorem guarantees that transmitting a signal from each antenna with the same measured phase will direct a beam back at the target. Applebaum [1] and Widrow [18] are examples of early work on adaptive arrays in the context of rejecting a jamming source from a phased array receiver.

Steinberg's Valley Forge Radio Camera [17, 15] used a standard adaptive method with a far-field point source to synchronize a large microwave random array. Once synchronized, the array's beam could be swept to image a target using standard phased array techniques. Later, Attia and Steinberg [2] worked to remove the need for a synchronizing source by using noncoherent reflections from the environment instead. These works all use a centralized beamforming architecture — all transmit/receive elements are tied into a single controller.

Barriac *et al.* [3] discuss the problem of distributed beamforming — removing the central controller. They note that the primary stumbling block is the distribution of

a common local oscillator signal (a signal whose phase fronts arrive at each element at the same time). They outline a master-slave architecture where each slave has an estimate of the path-delay between it and the master. Given some restrictions on the arrangement of the master and slaves, this is sufficient to simulate a common local oscillator at each slave. From there, the authors describe a reciprocity-based beamforming system quite similar to Steinberg's.

Separately, the same group propose an elegant gradient-ascent method [12] (*one-bit feedback*) that completely does away with the need for a common local oscillator. Only a common frequency source is required. It is an iterative process that requires an active receiver in the far field:

1. Each transmitter starts out with a random phase offset.
2. A random subset of transmitters change their phase.
3. The receiver indicates whether received power increased or decreased.
4. If received power decreased, the change from step 2 is reversed.
5. If received power has reached the required level, terminate.
6. Otherwise, return to step 2.

The end result of the process is a strong beam focused on the receiver. However, none of the individual transmitters have enough information for the beam to be steered in another direction.

One bit feedback methods have also been investigated for energy harvesting [11, 9]. In this framework, *energy receivers* send yes-or-no feedback to beamforming power transmitters, allowing effective power transmission without localization in noisy and cluttered environments.

Yong [19] and Jenn *et al.* [7] describe work with a proof-of-concept system for ship-borne phased array radar. In their design, all transmit and receive elements are wirelessly connected to a central controller. The rough position of each element is known, but compensation for small changes in path length between the controller and transmitters is required. The central controller measures the path delay to each transmitter in a round-robin fashion, and provides them with phase and magnitude compensation. Both Yong and Jenn note that the data transmission requirements of this architecture are immense.

Muralidharan and Mostofi [13] present an optimization framework to maximize received power at a base station from a set of mobile robots in a 2D workspace. They show that with certain assumptions, the problem can be approximated as an instance of the Multiple-Choice Knapsack Problem. Simulations of the optimization approach show good results. However, one of the assumptions they make is that the power contributed by multiple elements in purely additive — that no destructive interference can occur.

Ochiai *et al.* [14], followed by Buchanan and Huff [5, 6], present thorough analysis of the expected patterns of uniformly distributed random arrays of various shapes. In addition, Ochiai *et al.* present a clear overview of the dynamic beamforming problem, and the distinction between closed-loop and open-loop approaches to derivation of each element's phase information.

Recent work by Jensen [8] at Texas A&M University focused on solving the beamforming problem in a separate way. Jensen's MEDUSA system consists of 32 independently-positionable and orientable antennas with attached phase shifters. An RGBD camera focuses on the array, detecting LED-based fiducials. Computer vision software is used to estimate the position of each element, and each element's phase is adjusted in near-real time as its position changes.

4. ANALYSIS OF DEACTIVATION STEERING

It is not immediately clear from the definition of deactivation steering that it will form a coherent beam. In addition, the reliance on achieving coherence by deactivation raises concern that any beam that is produced will be too weak to be useful.

In this chapter, I:

- Provide an algorithm for choosing a maximal interval ($[\phi_{\max}, \phi_{\max} + w)$);
- Explore the expected array factor of a deactivation array, and show how it varies with respect to the tolerable phase error w;
- Compare the expected array factor of a deactivation array to that of a classical random array;
- Prove a lower bound on the number of elements that a maximal interval of width w will contain.

4.1 Finding a Maximal Interval

If the positions and intrinsic phase offsets of all elements are known, a simple algorithm (Figure 4.1) can be used to compute a maximal interval.

`choose_maximal_interval` sweeps a candidate interval over the entire phase space, counting the number of elements that fall within it. The sweep is discretized using the observation that the number of captured elements only changes when the candidate interval ends on an element. After completion of the sweep, the candidate interval containing the most elements is returned.

```
# Returns a maximal phase interval [φ_max, φ_max + w)
#
# Inputs:
#   * b̂: A unit vector in the desired main beam direction
#   * k: The wave number (2π/λ)
#   * p: An array of the element position vectors
#   * φ_0: An array of the element intrinsic phases
#   * w: The tolerable phase error (in radians)
def choose_maximal_interval(b̂, k, p, φ_0, w):
    φ = [k b̂ · p_i + φ_0i for p_i, φ_0i in zip(p, φ_0)]
    max_count = −∞
    max_interval = None
    for φ_i in φ:
        cur_count = 0
        for φ_j in φ:
            if φ_j ∈ [φ_i, φ_i + w):
                cur_count += 1
        if cur_count > max_count:
            max_count = cur_count
            max_interval = [φ_i, φ_i + w)
    return max_interval
```

Figure 4.1: An algorithm to choose a maximal interval.

All numeric evaluation of the array factor under the deactivation steering policy (Sections 4.2 and 4.3) uses maximal intervals selected using `choose_maximal_interval`.

4.2 Expected Array Factor of Deactivation Steering

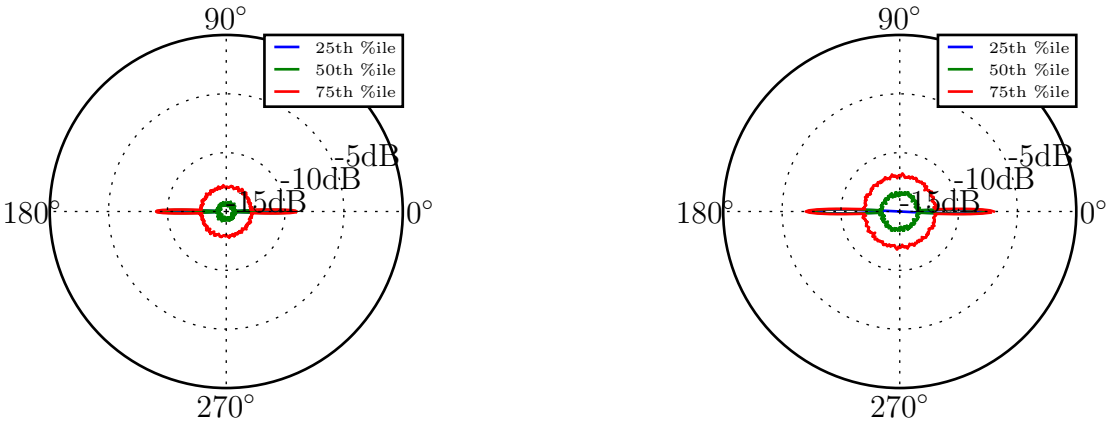
A random array, whether using classical steering or deactivation steering, uses elements whose positions are drawn from some underlying probability distribution. Since the array factor will change with each sampling of the distribution, it is more useful to characterize the steering strategy by its expected array factor:

$$\mathrm{E}\left[\mathrm{AF}(\hat{q})\right] = \sum_i \mathrm{E}\left[s_i \mathrm{e}^{-j(k\hat{q}\cdot\vec{p_i}+\phi_i)}\right] \tag{4.1}$$

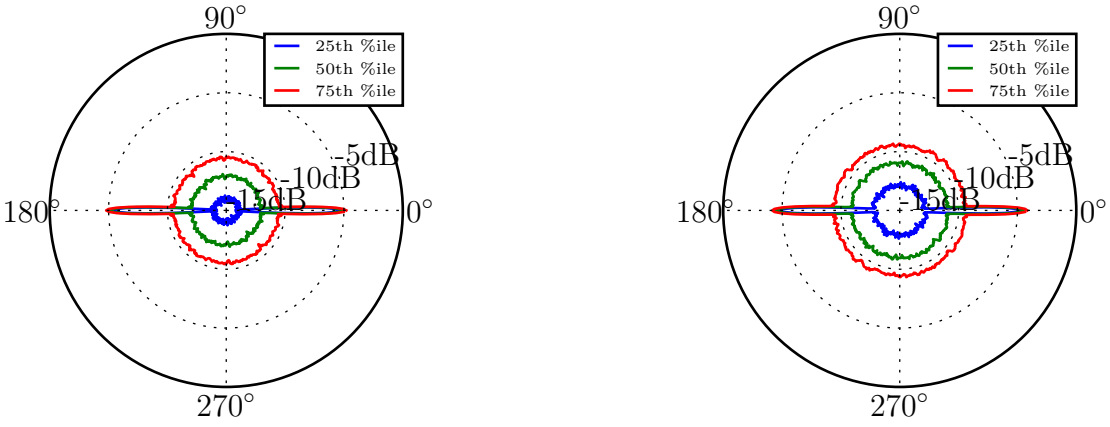
Analytic solutions for the expected array factor are quite complex, even for simple element distributions [5, 6]. Instead, I evaluate the expected array factor by collecting statistics over 1000 numeric trials.

Figure 4.2 plots the distribution of the array factor of a random array under deactivation steering. The particulars:

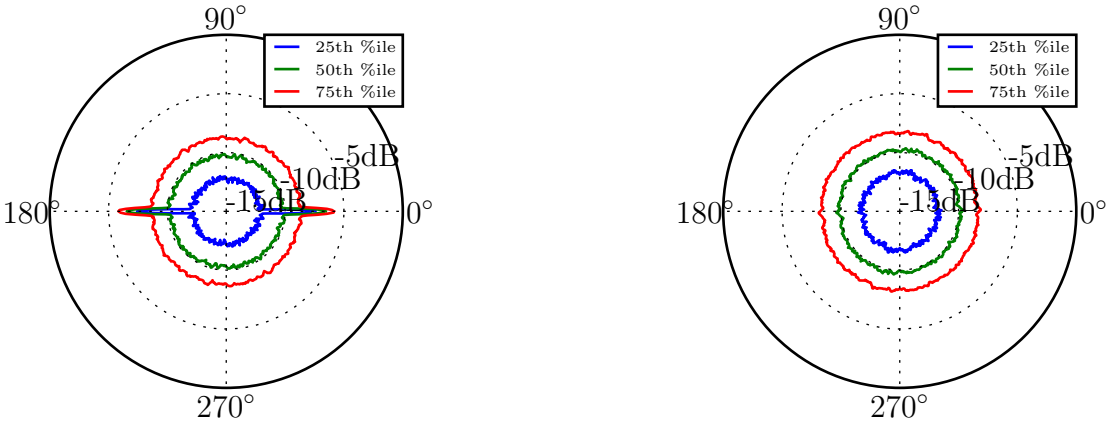
- The array consists of 64 elements;
- Element positions are drawn from a spherical uniform distribution with radius 10 m;
- The working wavelength of the problem is 1 m;
- The percentile statistics are calculated across 1000 trials — 1000 drawings of 64 elements from the underlying distribution;
- The elements are assumed to have intrinsic phase offsets $\phi_0 = 0$;
- The main beam vector $\hat{b}$ points along the $+x$ axis (0°, on the plots);



(a) $w = 5\% \cdot 2\pi$ (b) $w = 10\% \cdot 2\pi$



(c) $w = 25\% \cdot 2\pi$ (d) $w = 50\% \cdot 2\pi$



(e) $w = 75\% \cdot 2\pi$ (f) $w = 100\% \cdot 2\pi$

Figure 4.2: Array pattern under deactivation steering, $z = 0$ plane.

- The array factor is taken in the $z = 0$ plane;
- All power gains are taken relative to the main beam array factor of the same array under classical steering.

One prominent feature of deactivation steering is that the main beam sends power in both directions (0° and 180°, on the plots). This can be understood by analogy to a wavelength-spaced endfire array — deactivation steering groups its active elements into wavelength-spaced bands. Unlike a wavelength-spaced endfire array, the expected pattern of a deactivation-steered random array does not have to be precisely symmetric, since the distribution of active elements in phase space can be skewed.

As w is increased from 0 to 2π, the expected pattern changes shape. At $w = 5\% \cdot 2\pi$, the pattern as a whole is very weak (indeed, the pattern should decline to nothing at $w = 0$, since no elements will be active). At the other extreme, there is no main beam in the $w = 2\pi$ case. All elements are transmitting, so there is no directional preference.

When present, the main beam is well-distinguished from the sidelobes — even the 25th percentile of main beam gain is well above the 75th percentile of sidelobe gain. It should be noted that, although not plotted, the 0th percentile tends towards $-\infty$ and the 100th percentile tends towards the main beam power.

Figure 4.3 plots the dependence of main beam power on w in greater detail. The interquartile range is approximately 1 dB for most values of w, indicating that the expected main beam power density is reliable at any given w.

As w increases from 0 to 2π, the main beam gain quickly rises from $-\infty$, stays flat at approximately -5 dB from $25\% \cdot 2\pi$ to $75\% \cdot 2\pi$, and declines back to the sidelobe level at 2π. Median power density is maximized at approximately $w = 41\% \cdot 2\pi$, taking a gain value of approximately -4.4 dB.

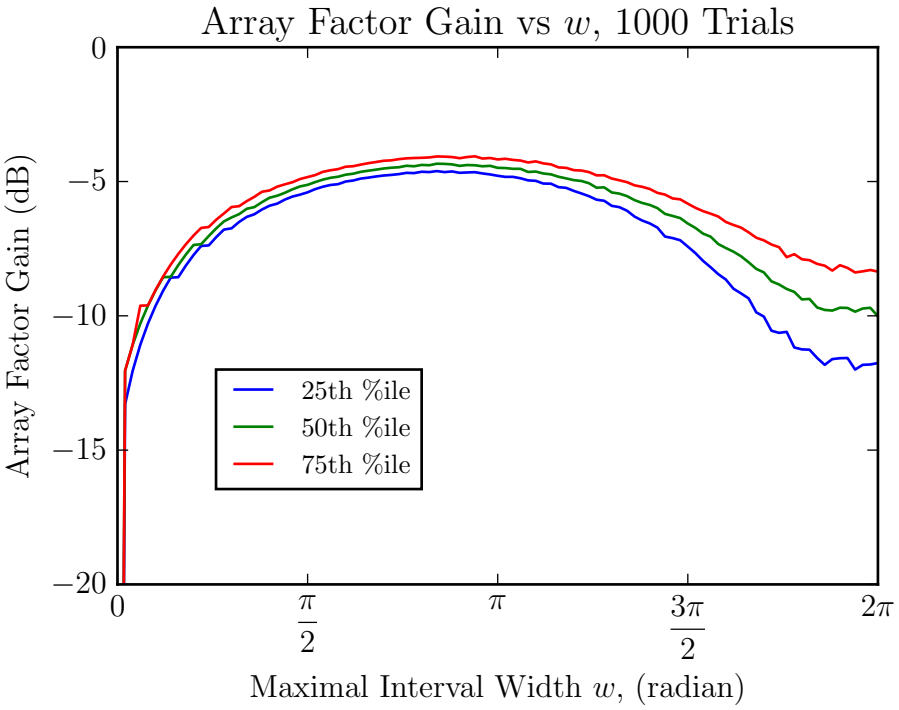


Figure 4.3: Array factor under deactivation steering versus w, sampled along the main beam ($\hat{q} = \hat{b} = \hat{x}$). Gain computed relative to the main beam power of a classically-steered array.

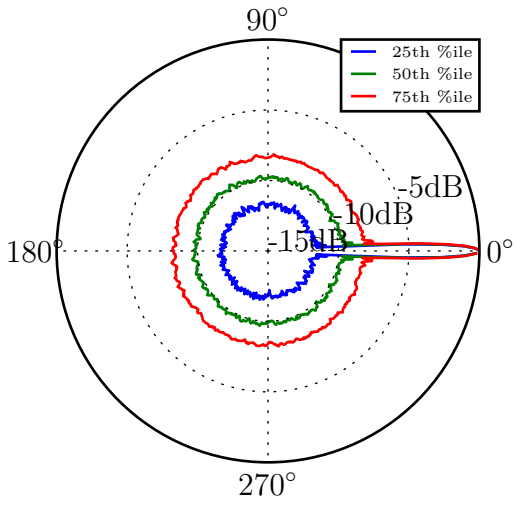


Figure 4.4: Array pattern under classical steering, $z = 0$ plane.

4.3 Comparison of Deactivation Steering and Classical Steering

The previous section's analysis shows that deactivation steering produces a clear main beam that is expected to be significantly stronger than the average sidelobe level. Now, I will compare the performance of deactivation steering and classical steering.

First, Figure 4.4 shows the array factor of a random array using classical steering. (The underlying parameters are the same as those used in the previous section — only the steering strategy has been changed.) The main beam is unidirectional, unlike the bidirectional main beam of deactivation steering, and the median sidelobe level is approximately 10 dB below the main beam.

Since deactivation steering is parameterized on w, I will choose w to maximize main beam power when comparing strategies. (Recall that main beam power under deactivation steering is maximized at approximately $w = 41\%2\pi$.)

Figure 4.5 shows the expected array pattern for this choice, and Figure 4.6 plots

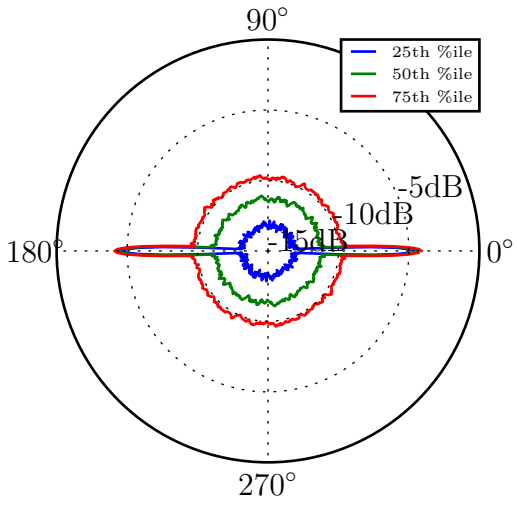


Figure 4.5: Array pattern under deactivation steering, $w = 41\% \cdot 2\pi$, $z = 0$ plane.

the median gain of both steering strategies near the main beam axis. On the main beam axis (0°, in Figure 4.6) there is a 4.38 dB difference between the steering strategies. In terms of direct power, this means that the deactivation-steered beam is roughly 36% as powerful as a classically-steered beam. Further analysis of the simulation reveals that half of this difference comes from the number of active elements. Deactivation steering leaves a median of 32 elements active, while classical steering always uses all 64 elements. The remainder of the difference comes from interference between elements on opposite sides of the active interval.

4.4 Element Availability Under Deactivation Steering

Given that deactivation steering deactivates all elements that lie outside a small section of phase space, concern over the reliability of the beam is warranted. Will there always be enough active elements to form an appreciable beam? How many active elements can be expected for a maximal interval width w?

First, there is a hard lower bound on the number of elements contained in a

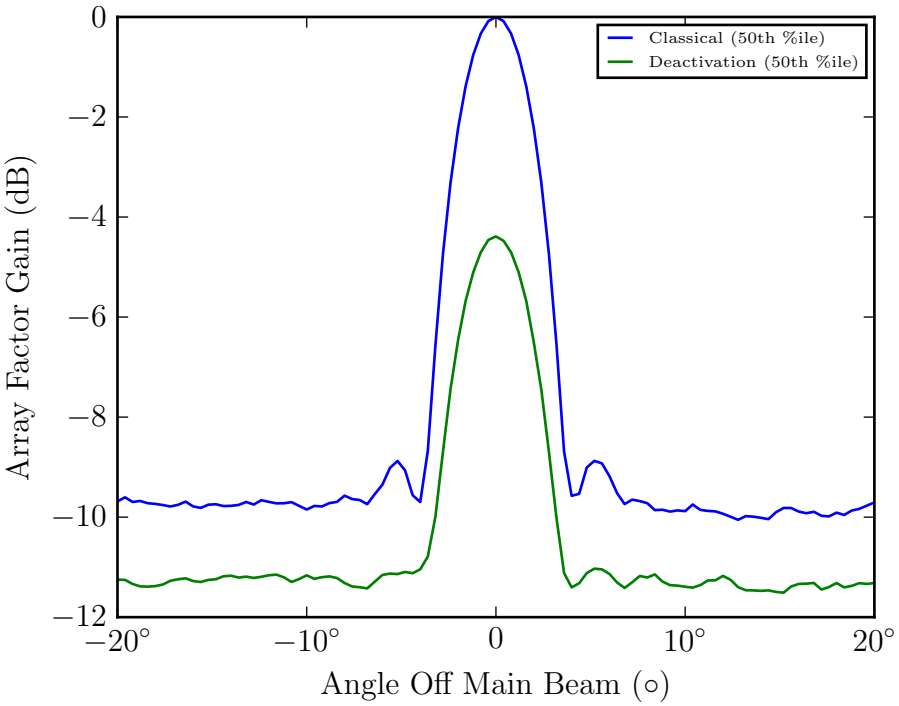


Figure 4.6: Array factor under deactivation ($w = 41\% \cdot 2\pi$) and classical steering within 20° of the main beam axis, $z = 0$ plane.

maximal phase interval of width w:

Lemma 1 (Extended Pigeonhole Principle)**.** *(Statement and proof adapted from [4], edited for clarity). If m items are to be divided among n bins, with the number of items in each bin denoted as m_i, then:*

$$\exists i : m_i \geq \left\lfloor \frac{m-1}{n} \right\rfloor + 1 \tag{4.2}$$

Proof. By contradiction. Assume the principle does not hold:

$$\forall i : m_i < \left\lfloor \frac{m-1}{n} \right\rfloor + 1 \tag{4.3}$$

$$\leq \frac{m-1}{n}. \tag{4.4}$$

This implies a contradiction:

$$m = \sum_{i=1}^{n} m_i \tag{4.5}$$

$$\leq \sum_{i=1}^{n} \frac{m-1}{n} \tag{4.6}$$

$$\leq n\frac{m-1}{n} \tag{4.7}$$

$$\leq m-1. \tag{4.8}$$

□

Theorem 1. *Given m elements distributed in phase space, it is always possible to choose a phase interval of width w that contains at least*

$$\left\lfloor \frac{m-1}{\lceil \frac{2\pi}{w} \rceil} \right\rfloor + 1 \tag{4.9}$$

elements.

Proof. Divide phase space into $\lceil \frac{2\pi}{w} \rceil$ phase bins, equally-sized and non-overlapping. Each of these bins has width

$$w_b \equiv \frac{1}{\lceil \frac{2\pi}{w} \rceil} \leq w. \tag{4.10}$$

Every element falls into one of these bins. Denote the number of elements in bin i as m_i. By Lemma 4.4, it is clear that:

$$\exists i : m_i \geq \left\lfloor \frac{m-1}{\lceil \frac{2\pi}{w} \rceil} \right\rfloor + 1. \tag{4.11}$$

Any interval of width w that completely covers this maximal bin necessarily contains at least the requisite number of elements. (See Figure 4.7). □

To get an intuitive sense of this lower bound, consider that most of the complication in its expression comes from properly handling integer boundaries. Approximately speaking, as m becomes large and w becomes small, it states that a maximal interval is guaranteed to contain at least $\frac{wm}{2\pi}$ elements.

The existence of this lower bound is independent of the element positions for any given array. This is a powerful guarantee — even if an adversary is moving elements to actively work against me, I will always be able to find a maximal interval containing an appreciable number of elements. I even still have free choice of w, allowing me to tune the beam power to my needs.

With this hard lower bound established, how many active elements can we expect in a non-adversarial case? Figure 4.8 plots both the guaranteed lower bound and the count of active elements (computed from the same element position distribution

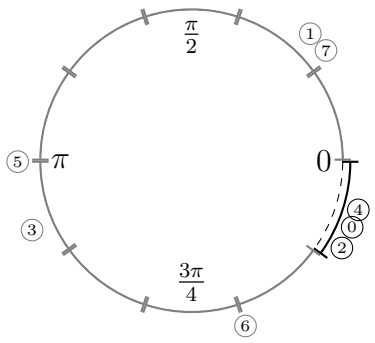


Figure 4.7: Pigeonhole principle in phase space. The black, outset phase bin is the bin that should covered with the active phase interval.

described in Section 4.2). From the plot, it is clear that deactivation consistently does better than the guaranteed bound, selecting approximately $\frac{wn}{2\pi}$ elements.

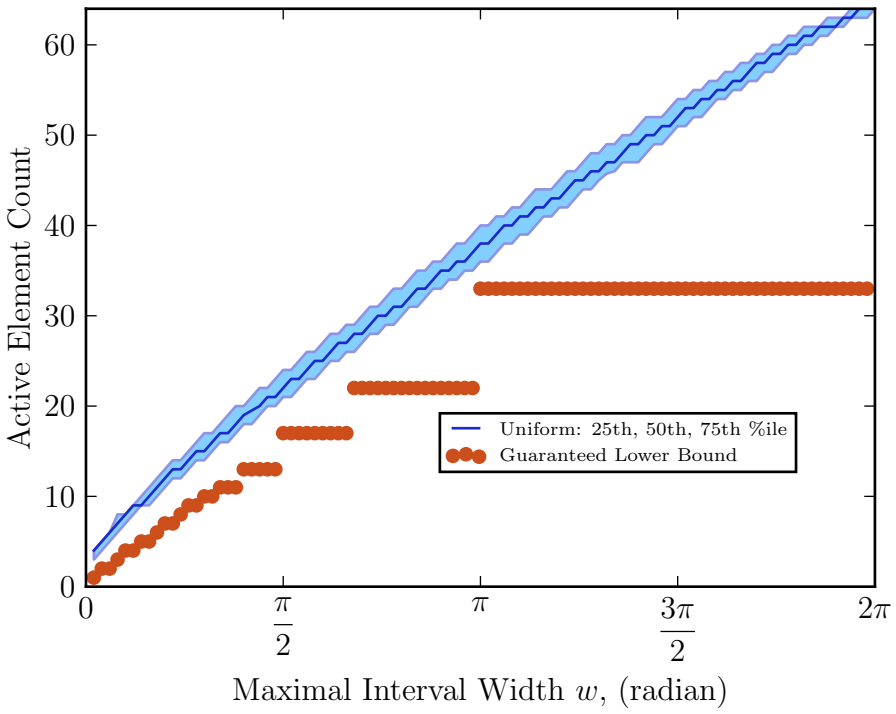


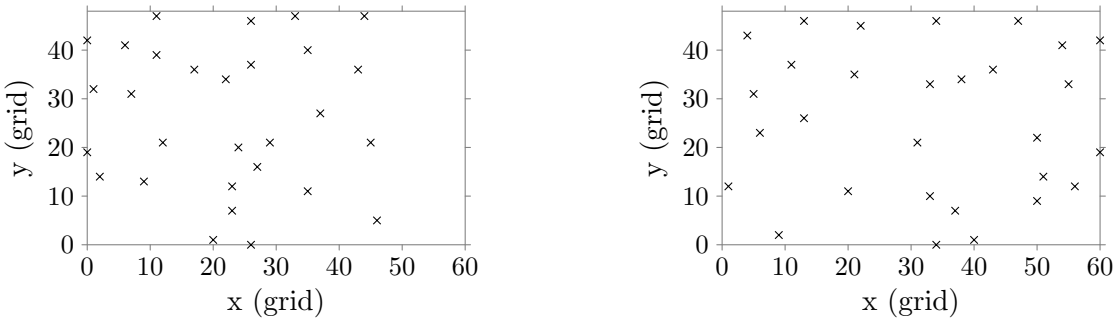
Figure 4.8: Simulated number of active elements, and the guaranteed minimum number of active elements, under deactivation steering.

5. BENCHTOP EXPERIMENTAL RESULTS

The previous chapter's analysis shows that deactivation steering is expected to form a well-defined beam, though less powerfully than classical steering.

To support that analysis, I ran a set of small-scale trials of deactivation steering. The trials were run at 2.5 GHz ($\lambda \approx 12$ cm), on a set of 27 monopole antennas. I ran two trials, each time choosing the element positions at random. The actual distribution of elements was not a simple uniform distribution: the optics table has a $60 \cdot 48$ grid of available positions on a 1 inch pitch, and each element has a small ground plane that prevents placing them too close together. The element positions for each trial were thus produced using a rejection approach: loop over all elements, repeatedly sampling a random position on the grid until a non-conflicting position is found. Figure 5.1 shows the resulting element positions for both trials.

For each trial, elements were fed using a 32-way power divider network connected to the active port of a network analyzer (Agilent FieldFox) (see Figures 5.2 and 5.3). Each leg of the power divider network introduces a separate, constant phase offset



(a) Trial 1 (b) Trial 2

Figure 5.1: Element positions. The main beam will be formed along the $+x$ axis (to the right).

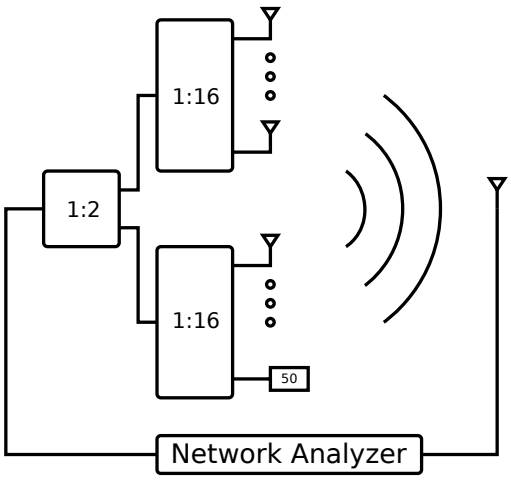
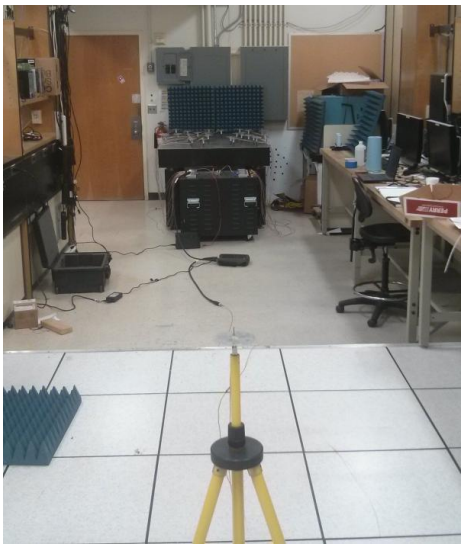


Figure 5.2: Block diagram of experiment trials.

(a) Closeup.



(b) Receiver.

Figure 5.3: The test setup. (a) shows the transmit antennas and power divider network. (b) shows the receiver antenna in the foreground, with the transmit array in the background.

between the input signal and the radiated signal for each element. These phase shifts were measured for each trial, and established as the intrinsic phase offset (ϕ_{0i}) for each element.

The input port of the network analyzer was connected to another monopole antenna. This receiver antenna was in the plane of the optics table, approximately 6.5 m away on the main beam axis.

Each trial followed these steps (starting with $n = 27$, $w = 2\pi$):

1. With n antennas active, measure the power gain between the two ports of the network analyzer.

2. Decrement n. If $n = 1$, stop. Otherwise, decrease w until the maximal interval (computed as in Section 4.1) contains n elements.

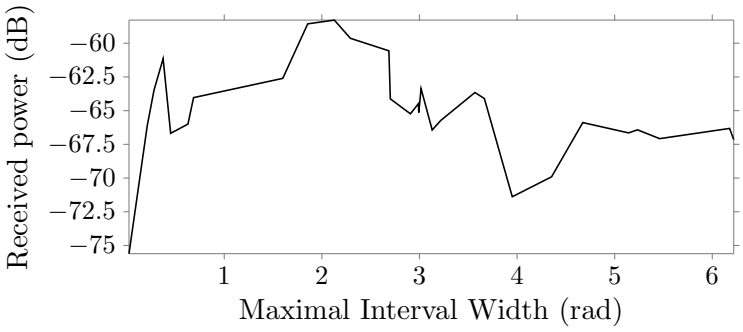
Figure 5.4 plots the resulting gain versus the maximal interval width.

The results of Trial 1 agree in general with my predictions in Sections 4.2 and 4.3. When w is small, received power is very low. As w increases, received power increases, attaining a maximum near $w = 2$rad, followed by a dropoff as w approaches 2π.

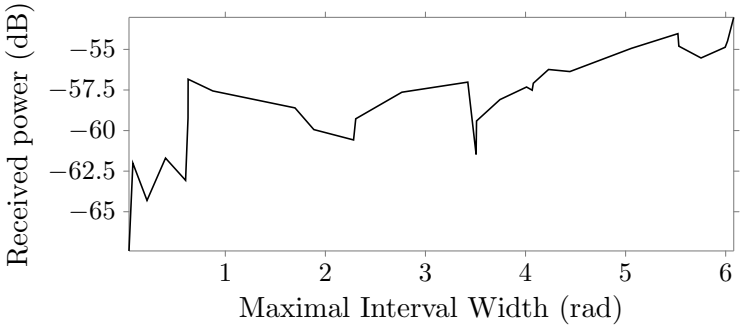
Trial 2 does not present a clear peak, and received power attains its maximum at $w = 2\pi$ (where all elements are active). This doesn't agree with my predictions, since I expect received power in the main beam to drop back to the sidelobe level as w approaches 2π.

The results from both trials are not very clean. There are two main factors that could contribute to this:

- *environmental reflections*: The lab environment in which the trials were run is electromagnetically cluttered. There are many large metal panels and wires that could reflect or scatter power from the sidelobes of the array to the receiver antenna.



(a) Trial 1



(b) Trial 2

Figure 5.4: Measured gain versus maximal interval width for both trials.

- *mutual coupling*: The optics table is quite crowded when filled with 27 transmitters — each trial had many elements within a single wavelength of each other. This causes some of the power emitted by an element to be captured and re-radiated by nearby elements. This effect can be mitigated by increasing the inter-element distance (physically, by increasing the diameter of the array; or virtually, by decreasing the working wavelength).

6. CONCLUSIONS

In this thesis, I proposed *deactivation steering*, a new method for controlling a random transmitter or receiver array. The primary benefit of deactivation steering is that it requires only passive knowledge of an element's phase space position, rather than active control — in other words, deactivation steering doesn't need phase shifters.

I provided an algorithm (`choose_maximal_interval`, Figure 4.1) that selects the best phase interval to activate for any given array, and analyzed the expected performance of arrays controlled by it. The expected behavior compares well to classically-steered random arrays, but pays an overall power penalty for deactivating a fraction of the available elements (forfeiting use of their output power). In addition, I proved that deactivation steering of width w is guaranteed to activate a certain fraction of the available elements, independent of their specific positions.

Finally, I conducted benchtop trials of random arrays using deactivation steering. These trials, while imperfect, indicate the basic viability of the steering method.

The analysis implies that deactivation steering is a useful control strategy that pays a power penalty compared to classical steering, but has the advantage of not requiring any phase shifters. There are situations where this may be a worthwhile tradeoff, for example:

- If an existing distributed system with many basic antennas must be retrofitted with dynamic beamforming capability;
- If cost analysis of a new system indicates that the price of per-element phase shifters outweighs the benefits of dynamic beamforming.

There are two immediate avenues for furthering this line of research. First, it would be helpful to run more physical trials of the system, with more elements, more widely separated. This will allow performing statistical tests of agreement between the simulated and actual distributions of main beam power. The second is to develop a distributed version of `choose_maximal_interval` — requiring a centralized algorithm for phasing limits the use of deactivation steering in decentralized environments, like robot swarms or wireless sensor networks. These decentralized, large-scale systems are likely to be where the particular tradeoffs of deactivation steering are most worthwhile.

REFERENCES

[1] S. Applebaum. "Adaptive arrays". In: *IEEE Transactions on Antennas and Propagation* 24.5 (Sept. 1976), pp. 585–598.

[2] E.H. Attia and B.D. Steinberg. "Self-Cohering Large Antenna Arrays Using The Spatial Correlation Properties of Radar Clutter". In: *IEEE Trans. Antennas and Propagation* AP-37.1 (Jan. 1989), pp. 30–38.

[3] G. Barriac, R. Mudumbai, and U. Madhow. "Distributed beamforming for information transfer in sensor networks". In: *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on.* Apr. 2004, pp. 81–88.

[4] A. Bogomolny. *Pigeonhole Principle and Extensions from Interactive Mathematics Miscellany and Puzzles.* URL: http://www.cut-the-knot.org/pigeonhole/PigeonholeExtensions.shtml (visited on 02/17/2017).

[5] K. Buchanan and G. H. Huff. "A comparison of geometrically bound random arrays in euclidean space". In: (July 2011), pp. 2008–2011.

[6] K. Buchanan and G. H. Huff. "A Stochastic Mathematical Framework for the Analysis of Spherically-Bound Random Arrays". In: *IEEE Transactions on Antennas and Propagation* 62.6 (June 2014), pp. 3002–3011.

[7] D. Jenn, M. Grahn, J.S. Gomez-Noris, R. Broadston, P. Djerf, et al. "Demonstration of a distributed digital phased array with wireless beamforming". In: *IEEE Antennas and Propagation Society International Symposium, 2008. AP-S 2008.* IEEE Antennas and Propagation Society International Symposium, 2008. AP-S 2008. July 2008, pp. 1–4.

[8] J. Jensen. “A Framework for Computer Vision Assisted Beamforming in Aperiodic Phased Arrays”. PhD thesis. Texas A&M University, 2015.

[9] S. Lee and R. Zhang. “Distributed energy beamforming with one-bit feedback”. In: *2016 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. Apr. 2016, pp. 398–403.

[10] Y.T. Lo. “A mathematical theory of antenna arrays with randomly spaced elements”. In: *IEEE Transactions on Antennas and Propagation* 12.3 (May 1964), pp. 257–268. ISSN: 0018-926X.

[11] D. Mishra, S. De, S. Jana, S. Basagni, K. Chowdhury, et al. “Smart RF energy harvesting communications: challenges and opportunities”. In: *IEEE Communications Magazine* 53.4 (Apr. 2015), pp. 70–78.

[12] R. Mudumbai, J. Hespanha, U. Madhow, and G. Barriac. “Scalable feedback control for distributed beamforming in sensor networks”. In: *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*. Sept. 2005, pp. 137–141.

[13] A. Muralidharan and Y. Mostofi. “Distributed beamforming using mobile robots”. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 6385–6389.

[14] H. Ochiai, P. Mitran, H. V. Poor, and V. Tarokh. “Collaborative beamforming for distributed wireless ad hoc sensor networks”. In: *IEEE Transactions on Signal Processing* 53.11 (Nov. 2005), pp. 4110–4124.

[15] B. Steinberg. “Radar imaging from a distorted array: The radio camera algorithm and experiments”. In: *IEEE Transactions on Antennas and Propagation* 29.5 (Sept. 1981), pp. 740–748.

[16] B. D. Steinberg. *Principles of Aperture and Array System Design: Including Random and Adaptive Arrays*. Jan. 1976. ISBN: 978-0471821021.

[17] B. D. Steinberg, E. N. Powers, D. Carlson, B. Meagher, R. S. Berkowitz, et al. "First experimental results from the Valley Forge radio camera program". In: *Proceedings of the IEEE* 67.9 (Sept. 1979), pp. 1370–1371.

[18] B. Widrow, P. E. Mantey, L. J. Griffiths, and B. B. Goode. "Adaptive antenna systems". In: *Proceedings of the IEEE* 55.12 (Dec. 1967), pp. 2143–2159.

[19] L. Yong. "Sensor synchronization, geolocation and wireless communication in a shipboard opportunistic array". MA thesis. Monterey, California: Naval Postgraduate School, Mar. 2006.

APPENDIX A

CLASSICAL STEERING SIMULATION

Python 3 source code for simulating array factor under classical steering:

```
import sys

import numpy as np
import scipy.constants as spc

def array_factor(emitter_position,
                 steering_vector,
                 sample_wave_vector):
    '''Compute the array factor for the given emitters'''
    steering_rank2 = np.reshape(steering_vector,
                                        (len(steering_vector), 1))

    sample_distance_phase = np.einsum('ik,jk->ij',
                                      emitter_position,
                                      sample_wave_vector)

    emitter_factor = steering_rank2 * np.exp(-1.0j * sample_distance_phase)

    # Sum the emitter factors over all emitters in each sample direction.
    return np.sum(emitter_factor, 0)

def run_trials(n_trials,
               sample_wave_vector,
               emitter_generator):
    '''Run N trials, returning array factors shape (N_TRIALS, N_SAMPLE_DIRS)'''
    trial_array_factor = np.empty((n_trials, sample_wave_vector.shape[0]))
    for trial_idx in range(n_trials):
        # Generate emitters for this trial
        emitter_position, steering_vector = emitter_generator()

        # Compute array factor for this trial
        af = array_factor(emitter_position, steering_vector, sample_wave_vector)
        trial_array_factor[trial_idx,:] = np.abs(af)

    return trial_array_factor

def classical_steering_func(emitter_position, main_beam_wave_vector):
    '''Compute the classical steering vector for the given emitters'''
    emitter_phase = np.einsum('ik,k->i',
                              emitter_position,
                              main_beam_wave_vector)
    return np.exp(1.0j * emitter_phase)
```

```
def main():
    '''Drive the simulation'''
    DIST_SIZE = 10

    N_EMITTERS = 64
    N_TRIALS = 1000

    N_SAMPLE_DIRS = 512

    MAIN_BEAM_DIR = np.asarray((1.0, 0.0, 0.0))

    WAVELENGTH = 1

    wave_number = (2 * spc.pi / WAVELENGTH)
    main_beam_wave_vector = wave_number * MAIN_BEAM_DIR

    # Generate a set of sample directions (confined to XY plane)
    sample_angle = np.linspace(0,
                               2*spc.pi,
                               N_SAMPLE_DIRS,
                               endpoint=False,
                               dtype=np.float32)
    sample_wave_vector = np.empty((len(sample_angle), 3))
    sample_wave_vector[:, 0] = wave_number * np.cos(sample_angle)
    sample_wave_vector[:, 1] = wave_number * np.sin(sample_angle)
    sample_wave_vector[:, 2] = wave_number * 0.0

    def dist_generator():

        points = np.empty((0,3))
        while points.shape[0] < N_EMITTERS:
            remaining = N_EMITTERS - points.shape[0]
            candidates = np.random.uniform(-1, 1, (remaining, 3))
            candidates = candidates[np.linalg.norm(candidates, axis=1) < 1]
            points = np.concatenate((points, candidates))
        emitter_position = points * DIST_SIZE
        steering_vector = classical_steering_func(emitter_position,
                                                  main_beam_wave_vector)
        return emitter_position, steering_vector

    trial_array_power = run_trials(N_TRIALS,
                                   sample_wave_vector,
                                   dist_generator)
    pctile_025_array_power = np.percentile(trial_array_power,  25, 0)
    pctile_050_array_power = np.percentile(trial_array_power,  50, 0)
    pctile_075_array_power = np.percentile(trial_array_power,  75, 0)

    # The array factor of a classically-steered random array along the main beam
    # is the number of elements in the array.
    ref_power = N_EMITTERS
```

```
    pctile_025_array_gain = 10 * np.log10(pctile_025_array_power / ref_power)
    pctile_050_array_gain = 10 * np.log10(pctile_050_array_power / ref_power)
    pctile_075_array_gain = 10 * np.log10(pctile_075_array_power / ref_power)

    # Output plot using matplotlib

    sys.stdout=sys.stdout.buffer

    import matplotlib as mpl
    pgf_with_custom_preamble = {'figure.autolayout': True,
                                "font.family": "serif",
                                "text.usetex": True,
                                "pgf.rcfonts": False,
                                "pgf.preamble": ["\\usepackage{siunitx}",
                                                 "\\usepackage{unicode-math}"]}
    mpl.rcParams.update(pgf_with_custom_preamble)

    import matplotlib.pyplot as plt
    plt.figure(figsize=(float(sys.argv[1]), float(sys.argv[2])))
    plt_pct025, = plt.polar(sample_angle,
                            pctile_025_array_gain,
                            label='25th %ile')
    plt_pct050, = plt.polar(sample_angle,
                            pctile_050_array_gain,
                            label='50th %ile')
    plt_pct075, = plt.polar(sample_angle,
                            pctile_075_array_gain,
                            label='75th %ile')
    plt.legend(handles=[plt_pct025, plt_pct050, plt_pct075],
               prop={'size': (10 if sys.argv[3] == 'svg' else 6)},
               bbox_to_anchor=(1.0, 1.0),
               borderaxespad=0)
    ax = plt.gca()
    ax.set_rlim([-15, 0])
    ax.set_yticks([-15, -10, -5])
    ax.set_yticklabels([r'-15dB', r'-10dB', r'-5dB'])
    ax.set_xticks([0, spc.pi/2, spc.pi, 3*spc.pi/2])
    ax.set_xticklabels(['$0^\circ$',
                        r'$90^\circ$',
                        r'$180^\circ$',
                        r'$270^\circ$'])
    plt.tight_layout()
    plt.savefig(sys.stdout, format=sys.argv[3], bbox_inches='tight')

if __name__ == '__main__':
    main()
```

APPENDIX B

DEACTIVATION STEERING SIMULATION

Python 3 source code for simulating array factor under deactivation steering:

```
import sys

import numpy as np
import scipy.constants as spc

def array_factor(emitter_position,
                 steering_vector,
                 sample_wave_vector):
    '''Compute the array factor for the given emitters'''
    steering_rank2 = np.reshape(steering_vector,
                                        (len(steering_vector), 1))

    sample_distance_phase = np.einsum('ik,jk->ij',
                                      emitter_position,
                                      sample_wave_vector)

    emitter_factor = steering_rank2 * np.exp(-1.0j * sample_distance_phase)

    # Sum the emitter factors over all emitters in each sample direction.
    return np.sum(emitter_factor, 0)

def run_trials(n_trials,
               sample_wave_vector,
               emitter_generator):
    '''Run N trials, returning array factors shape (N_TRIALS, N_SAMPLE_DIRS)'''
    trial_array_factor = np.empty((n_trials, sample_wave_vector.shape[0]))
    for trial_idx in range(n_trials):
        # Generate emitters for this trial
        emitter_position, steering_vector = emitter_generator()

        # Compute array factor for this trial
        af = array_factor(emitter_position, steering_vector, sample_wave_vector)
        trial_array_factor[trial_idx,:] = np.abs(af)

    return trial_array_factor

def deactivation_steering_func(emitter_position,
                               active_width,
                               main_beam_wave_vector):
    '''Compute deactivation steering vector for the given emitters'''
    ideal_phase = np.fmod(np.einsum('ik,k->i',
                                    emitter_position,
```

```
                                    main_beam_wave_vector),
                            2 * spc.pi)

    # Sweep a window of ACTIVE_WIDTH over all elements, remembering the start of
    # the window that contained the most elements.
    max_count = -1
    max_src = -1
    max_lim = -1
    for phase_i in ideal_phase:
        cur_count = 0
        window_src = phase_i
        window_lim = phase_i + active_width
        for phase_j in ideal_phase:
            if ((window_src <= phase_j and phase_j < window_lim)
                or (window_src <= phase_j + 2 * spc.pi
                    and phase_j + 2 * spc.pi < window_lim)):
                cur_count += 1
        if cur_count > max_count:
            max_count = cur_count
            max_src = window_src
            max_lim = window_lim

    gating = np.logical_or(np.logical_and(max_src <= ideal_phase,
                                          ideal_phase < max_lim),
                           np.logical_and(max_src <= ideal_phase + 2*spc.pi,
                                          ideal_phase + 2*spc.pi < max_lim))
    return gating.astype(np.complex128)

def main():
    '''Drive the simulation'''
    DIST_SIZE = 10

    N_EMITTERS = 64
    N_TRIALS = 1000

    N_SAMPLE_DIRS = 512

    MAIN_BEAM_DIR = np.asarray((1.0, 0.0, 0.0))

    WAVELENGTH = 1

    MAXIMAL_INTERVAL_WIDTH = float(sys.argv[1]) * 2 * spc.pi

    wave_number = (2 * spc.pi / WAVELENGTH)
    main_beam_wave_vector = wave_number * MAIN_BEAM_DIR

    # Generate a set of sample directions (confined to XY plane)
    sample_angle = np.linspace(0,
                               2*spc.pi,
                               N_SAMPLE_DIRS,
                               endpoint=False,
```

```
                          dtype=np.float32)
sample_wave_vector = np.empty((len(sample_angle), 3))
sample_wave_vector[:, 0] = wave_number * np.cos(sample_angle)
sample_wave_vector[:, 1] = wave_number * np.sin(sample_angle)
sample_wave_vector[:, 2] = wave_number * 0.0

def dist_generator():
    points = np.empty((0,3))
    while points.shape[0] < N_EMITTERS:
        remaining = N_EMITTERS - points.shape[0]
        candidates = np.random.uniform(-1, 1, (remaining, 3))
        candidates = candidates[np.linalg.norm(candidates, axis=1) < 1]
        points = np.concatenate((points, candidates))
    emitter_position = points * DIST_SIZE
    steering_vector = deactivation_steering_func(emitter_position,
                                                 MAXIMAL_INTERVAL_WIDTH,
                                                 main_beam_wave_vector)
    return emitter_position, steering_vector

trial_array_power = run_trials(N_TRIALS,
                               sample_wave_vector,
                               dist_generator)
pctile_025_array_power = np.percentile(trial_array_power,  25, 0)
pctile_050_array_power = np.percentile(trial_array_power,  50, 0)
pctile_075_array_power = np.percentile(trial_array_power,  75, 0)

# The array factor of a classically-steered random array along the main beam
# is the number of elements in the array.
ref_power = N_EMITTERS

pctile_025_array_gain = 10 * np.log10(pctile_025_array_power / ref_power)
pctile_050_array_gain = 10 * np.log10(pctile_050_array_power / ref_power)
pctile_075_array_gain = 10 * np.log10(pctile_075_array_power / ref_power)

# Output plot using matplotlib

sys.stdout=sys.stdout.buffer

import matplotlib as mpl
pgf_with_custom_preamble = {'figure.autolayout': True,
                            "font.family": "serif",
                            "text.usetex": True,
                            "pgf.rcfonts": False,
                            "pgf.preamble": ["\\usepackage{siunitx}",
                                             "\\usepackage{unicode-math}"]}
mpl.rcParams.update(pgf_with_custom_preamble)

import matplotlib.pyplot as plt
plt.figure(figsize=(float(sys.argv[2]), float(sys.argv[3])))
plt_pct025, = plt.polar(sample_angle,
                        pctile_025_array_gain,
```

```
                            label='25th %ile')
    plt_pct050, = plt.polar(sample_angle,
                            pctile_050_array_gain,
                            label='50th %ile')
    plt_pct075, = plt.polar(sample_angle,
                            pctile_075_array_gain,
                            label='75th %ile')
    plt.legend(handles=[plt_pct025, plt_pct050, plt_pct075],
               prop={'size': (10 if sys.argv[4] == 'svg' else 6)},
               bbox_to_anchor=(1.0, 1.0),
               borderaxespad=0)
    ax = plt.gca()
    ax.set_rlim([-15, 0])
    ax.set_yticks([-15, -10, -5])
    ax.set_yticklabels([r'-15dB', r'-10dB', r'-5dB'])
    ax.set_xticks([0, spc.pi/2, spc.pi, 3*spc.pi/2])
    ax.set_xticklabels(['$0^\circ$',
                        r'$90^\circ$',
                        r'$180^\circ$',
                        r'$270^\circ$'])
    plt.tight_layout()
    plt.savefig(sys.stdout, format=sys.argv[4], bbox_inches='tight')

if __name__ == '__main__':
    main()
```