

483R Interdisciplinary Project
Signal Intelligence Sensor Project
Portfolio

Ryan Adolph
Michael Oswald
Micaylyn "MJ" Parker

Montana State University
Bozeman, Montana
May 5, 2026



CSCI-483R
Michael Oswald,
Micaylyn Parker,
Ryan Adolph
05 May 2026

CSCI-483R Interdiscip. Proj.
Signal Intelligence Sensor
Portfolio

Signal Intelligence Scanner Portfolio

Portfolio

Contents

1 Abstract **4**

2 Introduction **4**

3 Qualifications **4**

4 Background **9**
4.1 Tools . . . 9
4.2 Similar Projects . . . 9
4.3 The Difference . . . 10

5 Work Schedule **10**
5.1 Responsibilities . . . 10
5.2 Gantt Chart . . . 12
5.3 Timeline . . . 15
5.4 Milestones . . . 16
5.5 Lifecycle Approach . . . 16

6 Project Technical Information **16**
6.1 Functional and Non-Functional Requirements . . . 16
6.2 Performance Requirements . . . 17
6.3 Interface Requirements . . . 17
6.4 Development Standards, Tools Used, etc. . . . 17

7 Methodology **18**
7.1 UML Diagrams . . . 20

7.2 Other Relevant Diagrams . . . 21
7.3 Design Patterns . . . 22
7.3.1 Pipeline Pattern (Pipes and Filters) . . . 23
7.3.2 Adapter Pattern . . . 23
7.3.3 Facade Pattern . . . 24
7.3.4 Strategy Pattern . . . 24
7.3.5 Producer–Consumer Pattern . . . 24
7.3.6 Layered Architecture . . . 25

8 Results **26**

9 Opportunities for Expansion **26**

10 Conclusion **26**

11 Works Cited **27**

12 Appendix **28**
12.1 User Interface . . . 28
12.1.1 Graphs.html . . . 28
12.1.2 Logs.html . . . 32
12.1.3 Radar.html . . . 34
12.1.4 SigIntParser.lua . . . 39
12.2 Main Processing Pipeline Script . . . 41
12.3 Beamforming Script (Python) . . . 42
12.4 Github . . . 45

1 Abstract

Spatial awareness and signal profiling are increasingly critical in both defense and civilian applications. This project explores the design and implementation of a modular, real-time audio sensing and processing system to determine the direction and possible identity of an acoustic profile using a Raspberry Pi platform. This system will use a common audio interface and multichannel microphone signal chain via USB connection, as well as a physical connection to an operator pc to provide rapid cycling of audio information and useful processing in an adaptive environment. This system will cycle through rapid sequences of audio acquisition, decomposition, beamforming, and FFT reduction to rapidly provide parsable spatial information based on the available microphone array. The 100-microphone array in this application is intended to be adaptable to what the user has available. To accomplish this, the system will be set up to accept continuous signal from any audio interface. To reduce the file size of these directional outputs, they will be processed by a Fast Fourier Transform (FFT) pipeline to extract frequency domain characteristics, stored as a text file on the Raspberry Pi, and waits for the operators PC to download the file. The operator's PC will then compare and contrast frequencies it has stored, being audio or radio frequencies, to produce a profile displayed on a basic application run on the operator's computer, showing the current signals profile. This system demonstrates a lightweight, modular, extensible framework for acoustic sensing suitable for deployment in resource-constrained and dynamic environments.

2 Introduction

The objective of this project is to design, implement, and demonstrate a functional signal intelligence system that combines observations of both the acoustic environment and RF spectrum to detect, identify, and characterize systems that emit signals it can observe.

With the combination of acoustic systems, FPGA and computational processing, RF receivers, and software-defined radio software, this project aims to create the building blocks of a detection system which is able to produce concise, actionable intelligence packets for broad use cases or future development. The system will be tailored towards accessible parts and ease of deployment, with an understandable, simple interface that reduces the full spectrum of information it receives to detailed information related to detected objects.

3 Qualifications

RYAN ADOLPH

(406)633-1979 · ryanadolph7@gmail.com

EXPERIENCE

JANUARY 2026 – CURRENT

BARTENDER, Tutti Benne – Bozeman, MT

Delivered high-end bar service within a fine dining experience, preparing craft and classic cocktails while also maintaining extensive knowledge of spirits and wines. Helped mentor and train new employees on service procedures while maintaining cleanliness and professionalism. Demonstrated creativity and adaptability through creating new cocktails during a fast-paced service.

FEBRUARY 2024 – OCTOBER 2025

BARTENDER / Server, est. 1864 Noir Bar & Restaurant

Provided a high level of service within an upscale environment. Mentored and trained employees on new service standards and procedures. Developed personal and professional connections with guests and staff through professional conversation. Demonstrated organization, creativity, and adaptability through a fast-paced and unpredictable environment.

EDUCATION

AUGUST 2020 - PRESENT

MAJOR- COMPUTER SCIENCE, MONTANA STATE UNIVERSITY

MINOR – COMPUTER ENGINEERING, MONTANA STATE UNIVERSITY

Expected Graduation: **May 2026**

Relevant Projects

- 8-bit Computer System on a DE10-Lite FPGA Board using VHDL
- Assembly and Embedded C programming experience on a TI MSP-430FR2355 and 430FR2310 Launchpad
- Multidisciplinary RC Car using an Arduino Mega 2560 Rev 3

SKILLS

- Knowledgeable in the command line
- Proficient in Java, VHDL, C, Assembly and Python
- Excellent debugging skills
- Willingness to learn about new hardware / software
- Excels in communication within a team

REFERENCES

- Kyle Crelin, Manager, Tutti Bene | (770) 865 - 4953
- Luke Greenwood, Owner, est. 1864 Noir Bar & Restaurant | (530) 310 - 9169
- Allison Fasano, Owner, est. 1864 Noir Bar & Restaurant | (347) 784-4870

Michael Oswald

106 Grant Chamberlain 3G
Bozeman, Montana, 59715
(406) 696-3340
oswald.contact@gmail.com

Education

B.S. Computer Science, Minor in Computer Engineering
B.A. Music Technology (3 semesters) T.
Montana State University, Bozeman, MT

Work Experience

Night Auditor | Field & Stream Lodge Co.
May 2025 → Present

- Guest centered customer service including reservations, maintenance issues, billing, and local experience information
- Night Audit, keeping and correcting books and other clerical matters

Private Event and Production Manager | Audio Engineer at The Pub Station
August 2022 → November 2023

- Client and event management for private events
- Manager and team leader for all events including security
- Audio and lighting tech including wireless system management
- Extreme situation management

Owner/Operator of Bród Sound LLC
October 2019 → 2023
(Freelance Audio Engineer | October 2015 → October 2019)
A formal platform for my services in audio; now operate as Bród Sound LLC.

- Provide, manage, and repair equipment for use and rental
- Live sound reinforcement
- Audio editing/repair/mixing
- Hand built microphones and cables for the best sound and cleanest stage

Allegra Marketing
August 2018 → March 2020
Wide format Print Technician providing large prints, sign making and installation, machining, and delivery driver

- Wide format printer maintenance and operation
- Media and substrate storage, ordering, and management
- Detailed quality control for sign image and construction
- Deliver products to customers in company vehicle
- On-site installation

Springhill Presbyterian Church | Bozeman United Methodist Church
January 2016 → November 2020 | May 2021 → September 2022
Lead site technician at a multi-site christian church.

- Video recording, playback, maintenance and troubleshooting
- Lighting control and maintenance
- Sound reinforcement/mixing, equipment support and troubleshooting
- Prudent time and talent management

Technical Skills

Operating Systems

- Windows
- OSX
- Linux

Platforms

- Pro Tools
- Reaper
- Logic
- Izotope RX
- Adobe Indesign
- Adobe Illustrator
- Adobe Photoshop
- General POS software
- Office Suite software

Electronic Tools/Experience

- Arduino
- Soldering
- Oscilloscope
- Digital multimeters
- Circuit analysis
- Circuit design
- Microphone construction
- General Electronics Repair

Other Skills

- Power tools
- Woodworking
- Machine operation
- Fast, reliable delivery

Programming languages

- C
- Python
- VHDL
- Assembly
- Java

General Skills

- Adaptable
- Detail oriented
- Leadership experience
- Customer service

Micaylyn J. Parker
micaylynparker@gmail.com

Education:

Montana State University, Major in Computer Science, Focus in Computer Engineering (current Senior)
Anne Arundel Community College, concentrations in Computer Science and General Engineering

Self-Development and Training:

Fundamentals of Software Engineering, Jan. 2020
Mathematics For SIGINT Exam, Jan. 2020
NSA Way Defect Tracking and Reporting, Jan. 2020
Introduction to Quantum Machine Learning with Xanadu and PennyLane, 2024

Work Experience:

September 2025 - Present Student Security Staff, Montana State University Applied Research Laboratory

Monitor facility access by verifying credentials, checking visitor badges, and tracking entry logs.	Maintain cleanliness and organization of front desk and common spaces to support a safe workspace.
Conduct routine inspections to confirm doors, equipment, and restricted areas are properly secured.	Greet and assist visitors, researchers, and staff while maintaining a professional and secure reception area.
Report and document any unusual activity, maintenance needs, or security concerns promptly to the Facility Security Officer.	Assist with access badge setup, log updates, and communication between staff and security management.

June 2022 - 2023 IT Support Specialist, Ascend Analytics, Bozeman, MT

Assist with the onboarding of new employees and user training on PC applications.	Upgrade software components and install OS upgrades as required.
Identify and resolve PC hardware and software-related failures and issues.	Update inventory of PC's and manage software updates to remote PC's.
Set up, configure, and maintain Windows laptops.	Assist in maintaining an Active Directory.
Handles tasks for project installs and helps define project tasks and task completion times.	Troubleshoot difficult problems to resolution quickly and to industry standard best practices.
Provide tier 1 Help Desk support- troubleshooting user concerns, application issues, and software installation.	Other responsibilities as assigned by the direct manager.

August 2020 - July 2022 Security Information Specialist, National Security Agency

Provide tier 1 Help Desk support- troubleshooting user concerns, application issues, and software installation.	Installed over 70 microcomputers at Polygraph Examiners' desks to allow access to Polygraph information at each desk.
Assist in maintaining an Active Directory.	Installing new workstations and monitors.

Maintaining and deploying polygraph examiner equipment.	Upgrading legacy technology, including installing new operating systems.
Installed over 70 ultra-wide monitors, allowing polygraph data to be viewed more easily.	Assuming additional responsibilities as the candidate's technological growth develops.
Facilitate cooperation, motivation, inspiration, and morale within the team.	Communicate information and ideas clearly, concisely, and professionally.
Help or assist coworkers or other employees when needed.	Upgraded Windows 7 and 8 PC workstations to Windows 10.
Work exceedingly well as part of a team.	Maintaining storage and supplies throughout the office.

2019 - 2020 High School Work Study, National Security Agency

Implemented processes to strengthen the organization by creating and managing the Events calendar. Allows the workforce to be aware of what is going on in the organization and throughout the agency.	Managed front office operations for leadership by maintaining a positive, customer-focused atmosphere as the first-line point of contact. Professionally greeted visitors and ensured accurate and complete messages were taken.
Primary calendar support for leadership and staff of twelve. Back-up support for the Executive Assistants, and ensured continuous office and admin support.	Responded to the call to provide backup to higher-up office managers, filling the gap to support the Leadership.
Cleaned and organized supplies in the office.	Helped reduce the average data input time for Vehicle Rejection sheets from 17 to 2 days.

Computer Skills:

● MS Office suite (Word, Excel, PowerPoint, Publisher, etc.) ● Adobe Suite (Illustrator, Photoshop, Acrobat Pro, InDesign, Premiere Pro, Lightroom, Audition, etc.) ● Java/C/Assembly programming	● Google Drive suite (Docs, Sheets, Slides, Forms, etc.) ● Autodesk Inventor ● MultiSIM ● GMail

Honors:

National Honor Society, Inducted 2018
National Spanish Honor Society, Inducted 2017

Extracurricular Activities:

Marching Band:

- 2022-Present Montana State University's Spirit of the West, Drumline
- 2015-2020 Severna Park High School, Percussion

Avid volunteer

4 Background

The modern environment is increasingly populated with both acoustic and electromagnetic emissions. These signals can be used to characterize important modern systems such as transportation, wireless communication, Electronic Counter Measures (ECM), and drones. These modern systems operate by using moving parts and wireless communications across numerous distinct bandwidths or noise profiles that make them trackable via these signatures.

Acoustic and signal detection and analysis have been explored and implemented constantly in various form factors with respect to cost, complexity, and application domain. The differences in cost, deployability, and stability have a significant impact on both civilian and defense applications. Systems and methods that allow sensory intelligence to be deployed easily, cheaply, and with parts that are and will continue to be available are immensely valuable in any application. Exploring and developing such a system is the goal of this team.

4.1 Tools

- Raspberry Pi 4 Model B
- Audio Interface (Mixpre 6 II)
- Microphones (4x Sure Beta 98)
- Connecting cables
- Operator PC

4.2 Similar Projects

- FPGA Implementation of a Bartlett Direction of Arrival algorithm for a 5.8GHz Circular Antenna Array, 2016
  - This paper outlines an FPGA implementation of a Direction of Arrival algorithm using an array of Antennas
  - This research was done locally by the Montana State Electrical and Computer Engineering Department by Brock J. LaMeres, Yikun Huang, Monther Abusultan, Sam Harkness
- Developing A Real-Time SDR System, Implementing software-defined radio by integrating a tool like GNU Radio with real FPGA hardware and design and verification tools. July 29th, 2021
  - This paper outlines how a Software-Defined Radio System (SDR) is implemented on a FPGA board.

  - The main goal of an SDR is to transfer audio data through analog channels, breaking down the incoming wave signal into digital signals.

4.3 The Difference

Our project was meant to be a system that everyday people would be able to purchase, set up, and upgrade on their own. The difference from the Bartlett DOA algorithm uses an FPGA, which is a soft microprocessor system, which is much faster at processing compared to our Raspberry Pi system. They also implemented their system using an antenna array, while our system uses 4 microphones and an audio mixer connected to a Raspberry Pi with a processing speed of 1.8 GHz, while the FPGA implementation can process up to 4 Gbps.

We wanted to keep our system as straight-forward as possible, meaning we didn't want to have to implement a SDR on our system, including the other elements being combined on this one project. Including an SDR that was already implemented by our audio mixer was the easiest solution to this problem. The microphones automatically plugged into an SDR that turned audio into a digital signal (.wav file) that our raspberry pi is able to read and analyze through a FFT.

5 Work Schedule

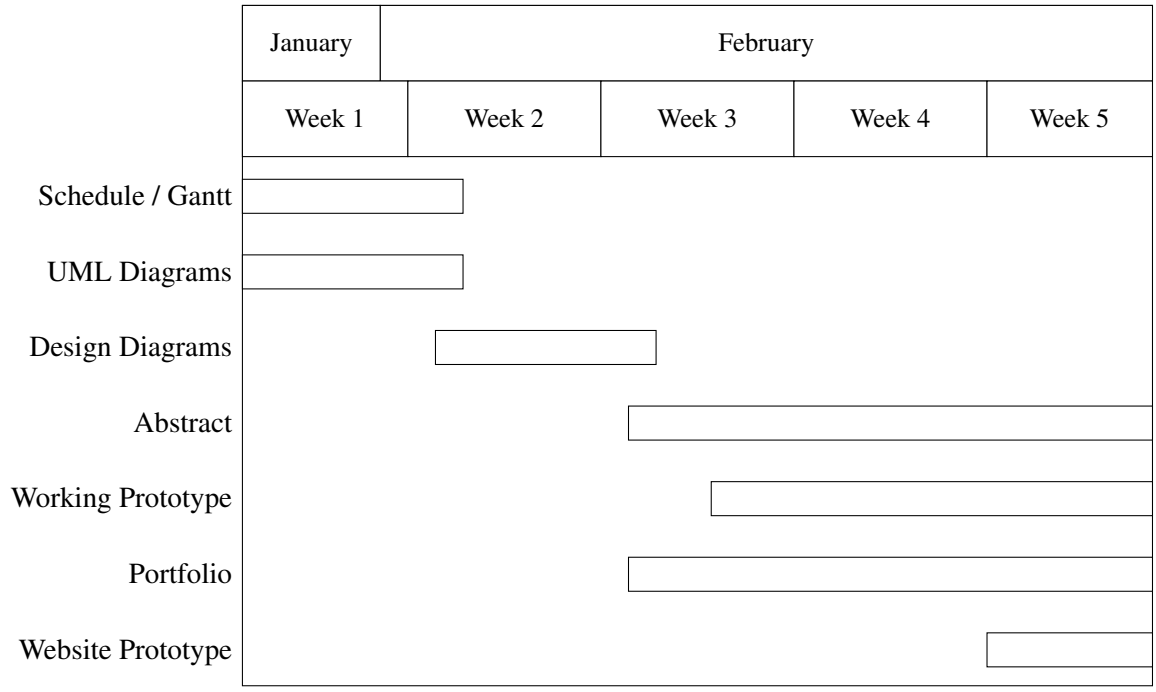
5.1 Responsibilities

- The whole group was responsible for:
  - Personal Resumes
  - Poster Completion
  - Portfolio Completion
  - Research Celebration Presentation (5/1/26)
  - Final Capstone Presentation (5/5/26)
- Ryan Adolph was responsible for:
  - Finding an open-source FFT (2/3/26 - 2/4/26)
  - Implementing FFT with our initial microphone and reworking to get more data out of FFT. (2/4/26 - 2/13/26)
- Michael Oswald was responsible for:
  - Project Topic

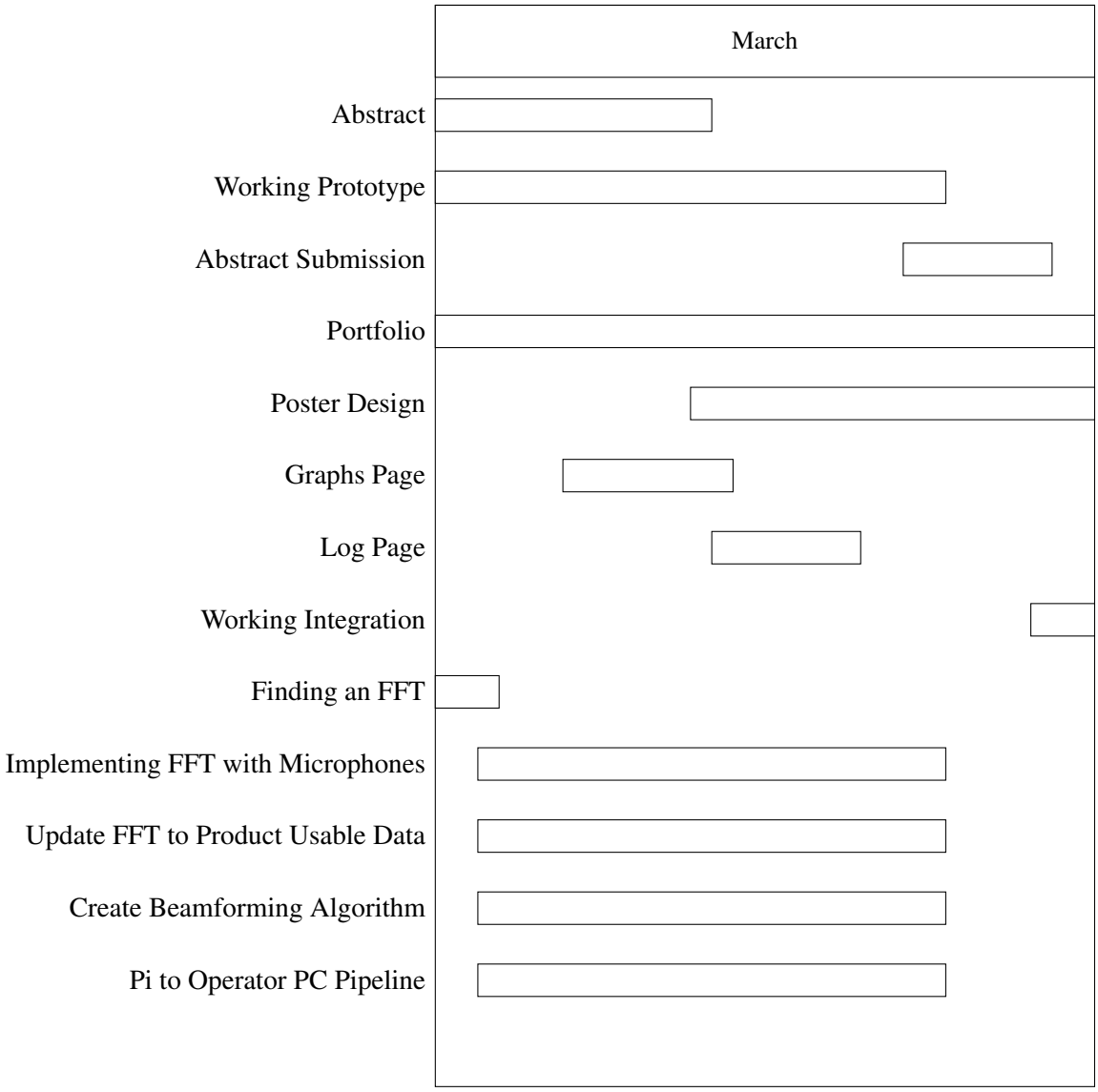
  - UML Diagrams
  - Pipeline Design
  - Audio Interface Integration
  - Portfolio
  - Poster design
- Micaylyn Parker was responsible for:
  - Completing the Work Schedule and Gantt Chart
  - Completing the Web App and User Interface
    - Graphs Page
    - Log Page
    - Radar Page
    - Implementation
  - Poster Design
  - Portfolio Creation and Design

5.2 Gantt Chart

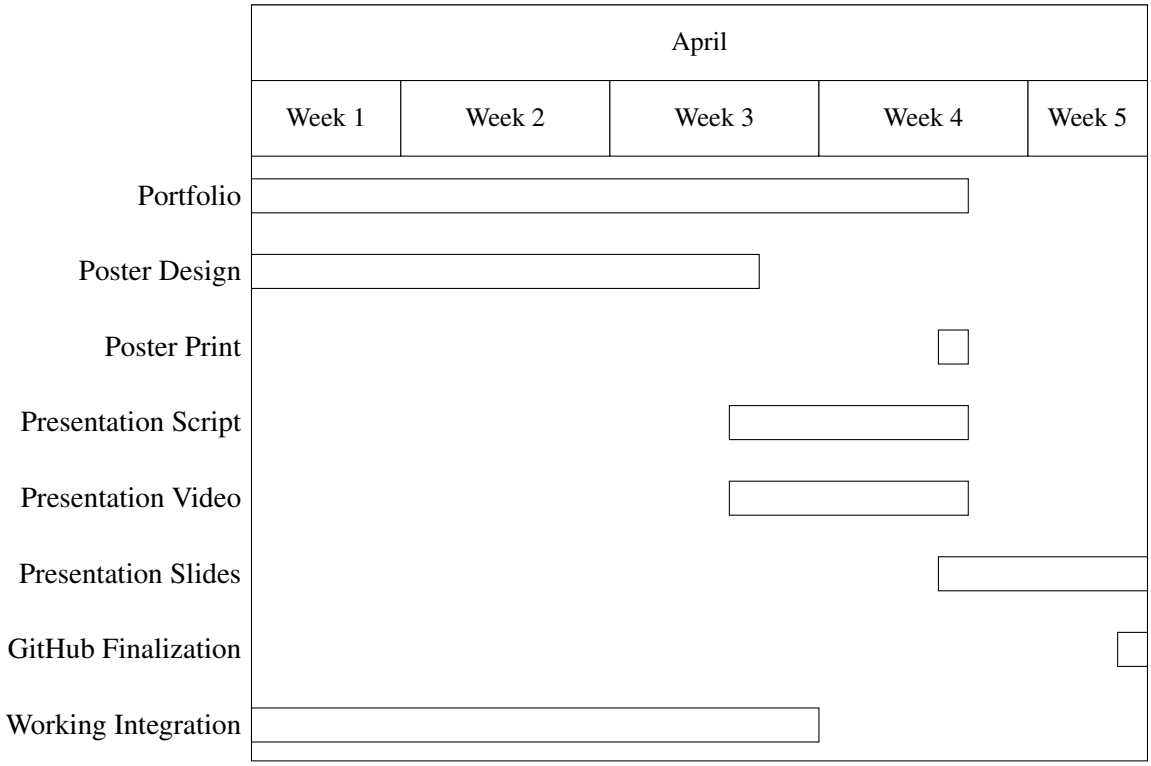
January – February 2026



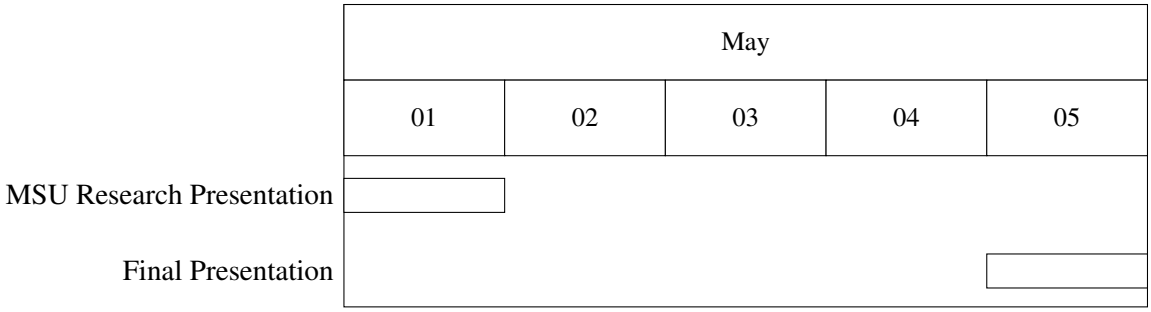
March 2026


March
Abstract
Working Prototype
Abstract Submission
Portfolio
Poster Design
Graphs Page
Log Page
Working Integration
Finding an FFT
Implementing FFT with Microphones
Update FFT to Product Usable Data
Create Beamforming Algorithm
Pi to Operator PC Pipeline

April 2026



May 2026



5.3 Timeline

Task	Start	End
Documentation		
Create Schedule / Gantt Chart	1/27/26	2/3/26
UML Diagrams	1/27/26	2/3/26
Design Diagrams	2/3/26	2/10/26
Abstract	2/10/26	3/13/26
Working Prototype	2/13/26	3/24/26
Abstract Submission	3/23/26	3/29/26
Portfolio	2/10/26	4/24/26
Poster Design	3/13/26	4/17/26
Poster Print	4/24/26	4/24/26
MSU Student Research Celebration Presentation	5/1/26	5/1/26
GitHub Finalization	4/30/26	4/30/26
Presentation Script	4/17/26	4/24/26
Presentation Slides	4/24/26	4/30/26
Presentation Video	4/17/26	4/24/26
Presentation	5/5/26	5/5/26
User Interface		
Basic Website Prototype	2/23/26	3/1/26
Login Page	3/1/26	3/7/26
Graphs Page	3/7/26	3/14/26
Log Page	3/14/26	3/20/26
Working Integration	3/29/26	4/19/26

Table 1: Project Schedule

5.4 Milestones

- Implementing the FFT with the microphone(s) (2/28/26)
- Creating a stand for the microphones
- Updating FFT code to produce more data to implement beamforming (3/4/26)
- User Interface design started (2/22/26)
- Created beamforming algorithm
- Raspberry Pi to Operator PC pipeline (3/23/26)
- User Interface was updated (4/28/26)
- Prototype testing (3/24/26)
- Student Research Celebration (5/1/26)
- Portfolio updated and finalized (5/4/26)

5.5 Lifecycle Approach

This product has the potential for much more expansion, given more time on the project. For example, implementing this system with a larger array of microphones could increase the range it is able to detect. The use of this product is to gather information we would not normally have at a much faster speed than a human can, and staying within a reasonable cost. There is a possibility for the system to be implemented on a military vehicle for defense against enemy vehicles or for it to be set up as a home system detecting intrusions and where someone may be in a building, given the microphones are powerful enough.

The system was designed with simplicity and ease in mind, so that any non-technical person is able to set this system up and use it.

6 Project Technical Information

6.1 Functional and Non-Functional Requirements

- Functional Requirements
  - Detect a signal being emitted
  - Profile a signal being detected
  - Match audio signal to RF signal

  - Consistently tracks object location and behavior
  - Create and log a database of object profiles for identification and comparison
- Non-Functional Requirements
  - Minimized the overhead of code/hardware needs
  - Avoid displaying noise in the final User Interface presentation
  - Display simple User Interface
  - System setup can be done by non-technical personnel / is simple

6.2 Performance Requirements

- Record 0.25-second recording samples every 0.5 seconds
- Store the audio file recorded
- Store FFT analysis data from each recording, matching result name to .wav file name

6.3 Interface Requirements

- Must display each microphones output
- Must use a layout that is easy to read and navigate.
- Must allow the user to view microphone output without additional configuration.
- Shall display a scroll-able system log containing recent events and errors.
- Shall include a timestamp for every entry within the system log.
- Shall display radar-detected targets in a real-time.

6.4 Development Standards, Tools Used, etc.

In order to ensure our accreditation for this course through Montana State University, we needed to maintain and follow our selected design pattern when developing our system, which is the Factory Design pattern. The goal was to keep our systems as modular as possible for future improvements and upgrades.

To start, we used an open-source FFT visualizer to run the FFT algorithm so we didn't have to write this from scratch. We edited this code to produce useful data to use during future analysis. We used an audio mixer that implemented noise filtering from the microphones before sending the .wav file to the Pi for FFT and beamforming analysis.

7 Methodology

A 4-microphone array captures audio in 0.25-second intervals every 0.50 seconds, which outputs one large .wav file filled with 4 audio inputs. We then split the large .wav file into 4 separate .wav files to perform a beam forming analysis to locate which direction the audio signal was coming from. We then perform an FFT over the 4 .wav files to analyze each frequency. We then store the FFT output for the operator's PC to come and grab to perform more signal analysis.

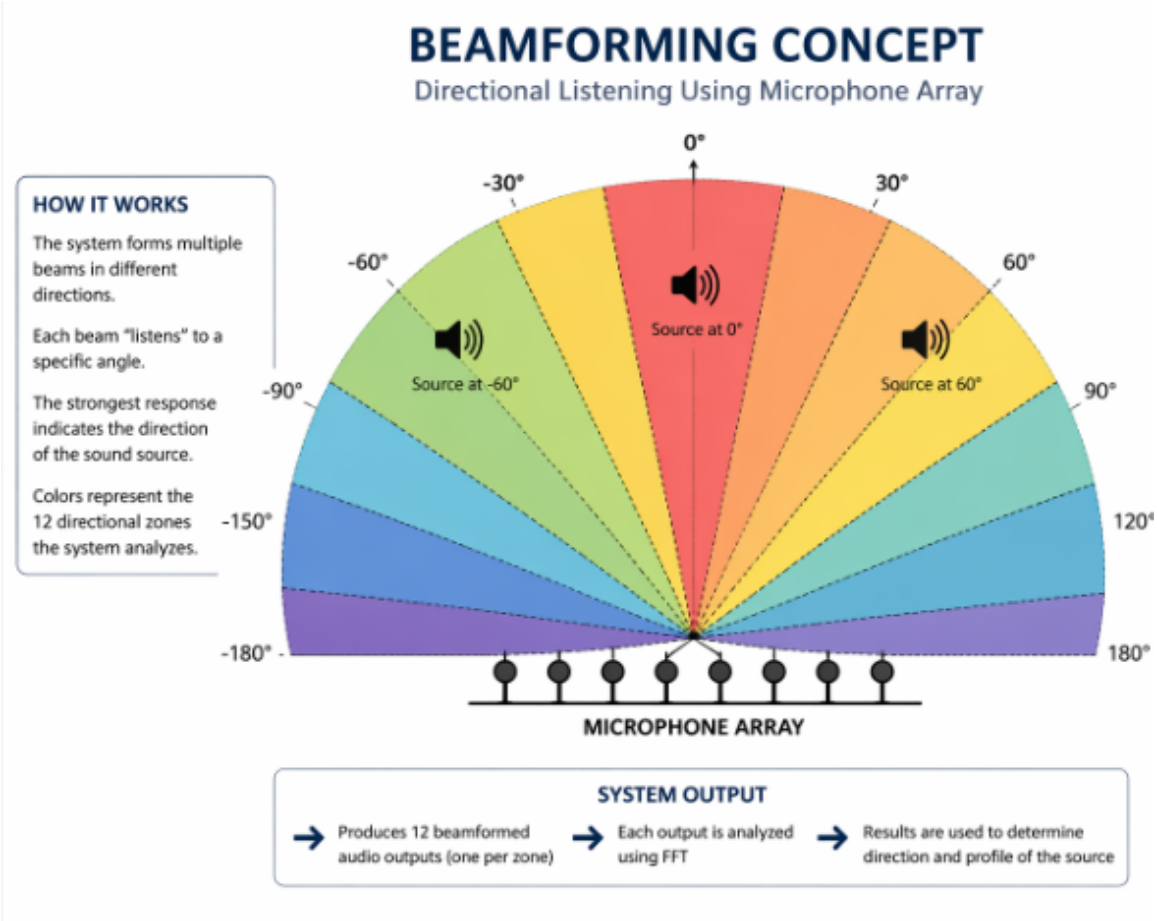


Figure 1: Beamforming Concept

As the complete system is developed, the hardware and integration requirements will be constantly re-evaluated to maintain the goal of accessible and scalable systems.

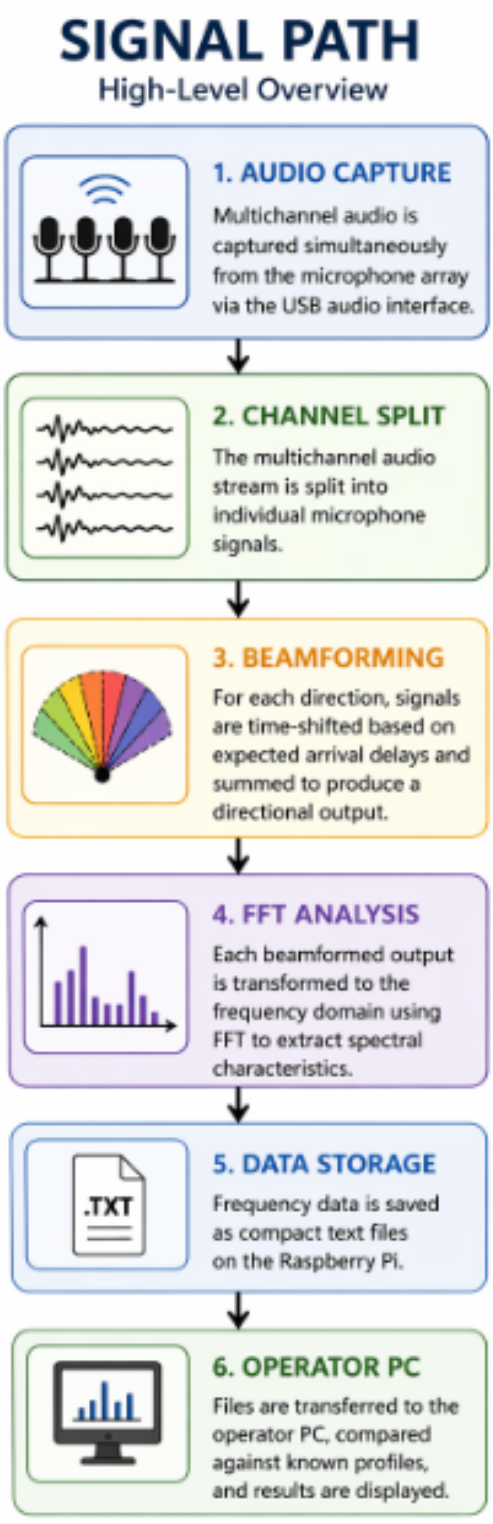


Figure 2: Signal Path

7.1 UML Diagrams

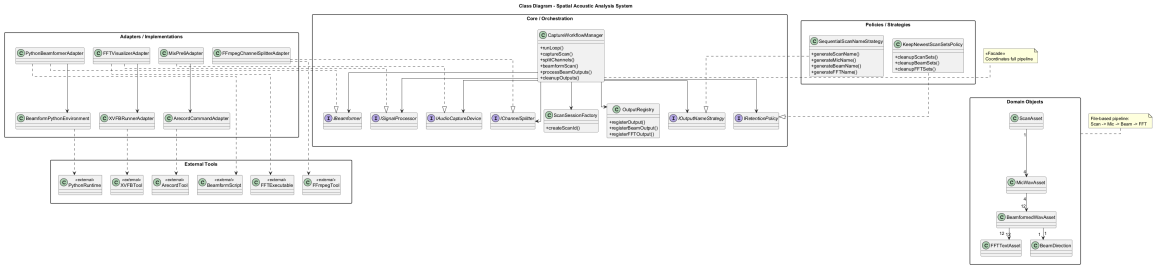


Figure 3: Class Diagram

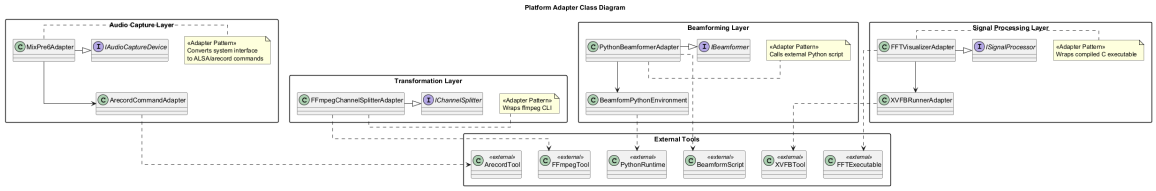


Figure 4: Adapter Class Diagram

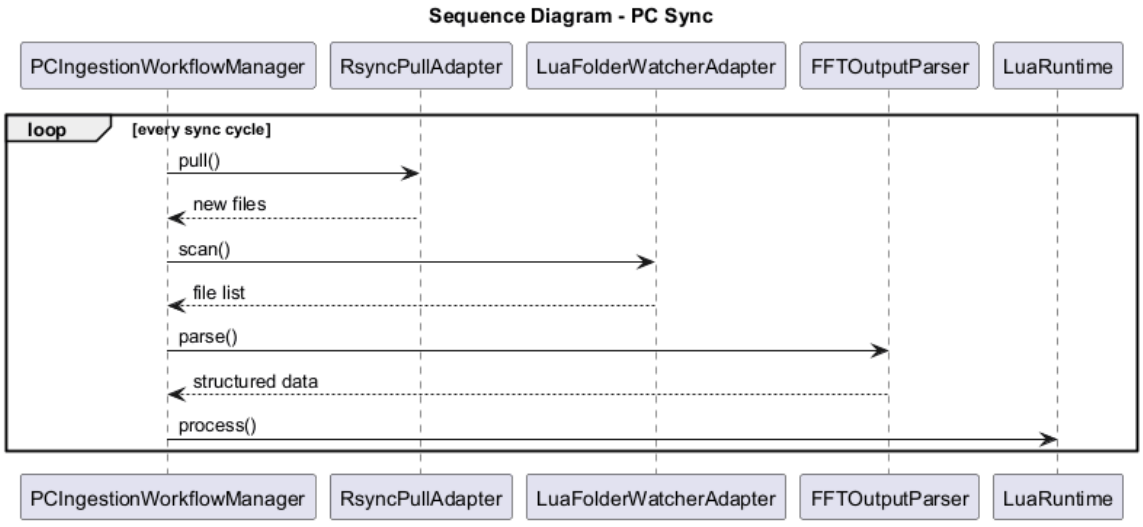


Figure 5: PC Sync Sequence Diagram

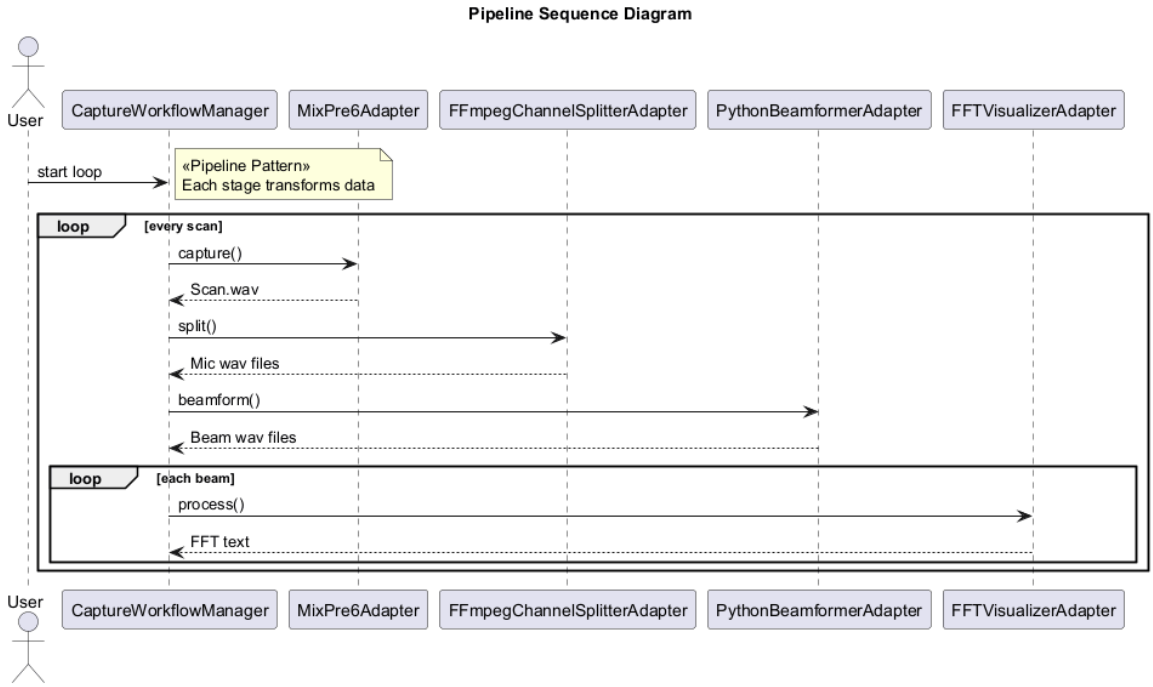


Figure 6: Pipeline Sequence Diagram

7.2 Other Relevant Diagrams

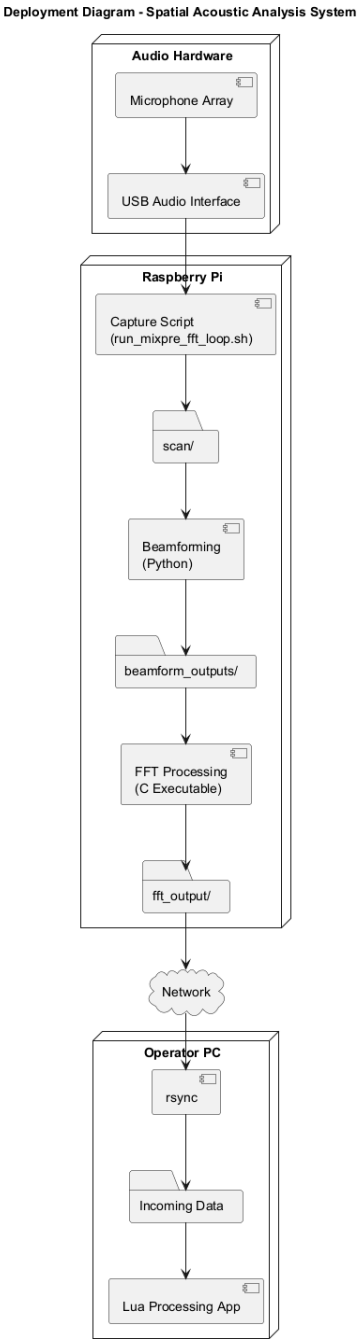


Figure 7: Deployment Diagram

7.3 Design Patterns

The system architecture reflects a deliberate selection of design patterns that support modularity, extensibility, and interoperability across heterogeneous tools and execution environments. Rather

than relying on a single architectural paradigm, the design combines multiple complementary patterns to address different aspects of the system's requirements, including real-time data processing, integration of external tools, and distributed analysis.

7.3.1 Pipeline Pattern (Pipes and Filters)

The primary architectural pattern implemented in this system is the Pipeline (Pipes and Filters) pattern. This pattern structures the system as a sequence of independent processing stages, where each stage performs a specific transformation on the data and passes the result to the next stage. The pipeline model promotes strong separation of concerns, enabling each processing step to be developed, tested, and modified independently.

In this implementation, the pipeline operates on audio data captured from a multichannel microphone array. The processing sequence consists of four primary stages: audio capture, channel decomposition, spatial beamforming, and frequency-domain analysis. These stages are executed in order, producing intermediate artifacts at each step. The data flow can be summarized as:

$$\text{Scan.wav} \rightarrow \text{Mic_*.wav} \rightarrow \text{Beam_Angle_*.wav} \rightarrow \text{FFT_Output.txt}$$

A key design decision was to implement the pipeline using file-based interfaces rather than in-memory data transfer. While this introduces some performance overhead, it enables seamless interoperability between tools written in different languages and running in different environments, including shell utilities, Python scripts, and compiled C executables. This approach also simplifies debugging, as each stage produces persistent outputs that can be inspected independently. The pipeline pattern therefore serves as the structural backbone of the system.

7.3.2 Adapter Pattern

To support integration of external tools and hardware interfaces, the system employs the Adapter pattern. This pattern allows components with incompatible interfaces to be used within a unified system by translating their native interfaces into a consistent internal abstraction.

In practice, this system integrates several external tools, including ALSA (`arecord`) for audio capture, `ffmpeg` for channel decomposition, a Python-based beamforming script, and a compiled C executable for FFT processing. Each of these tools has distinct invocation methods, parameter requirements, and output formats. Rather than embedding these details directly into the core workflow logic, the system treats each tool as an interchangeable processing component with clearly defined input and output contracts.

This abstraction isolates the system from tool-specific implementation details and enables future substitution of components. For example, the FFT processing stage could be replaced with a hardware-accelerated implementation on an FPGA without requiring changes to the surrounding pipeline. The Adapter pattern therefore plays a critical role in maintaining flexibility and portability.

7.3.3 Facade Pattern

The system also incorporates the Facade pattern to simplify interaction with a complex set of processing components. The Facade pattern provides a single, unified interface to a subsystem that would otherwise require coordination of multiple independent elements.

In this implementation, the shell script `run_mixpre_fft_loop.sh` serves as the facade for the entire system. This script orchestrates all stages of the pipeline, including audio capture, channel splitting, beamforming, FFT processing, output management, and data cleanup. From the perspective of the operator, the system can be executed and controlled through this single entry point, without requiring direct interaction with the underlying tools.

This design significantly improves usability and reduces operational complexity, particularly in deployment scenarios where reproducibility and reliability are critical. The Facade pattern therefore provides a practical abstraction layer that bridges system complexity and user interaction.

7.3.4 Strategy Pattern

The Strategy pattern is applied to encapsulate system policies that may vary independently of the core processing logic. This pattern allows different behaviors to be defined and substituted without modifying the main workflow.

In this system, strategy-like behavior is implemented in the handling of output naming conventions and data retention policies. Sequential scan naming ensures that output files are organized in a predictable and traceable manner, while retention logic enforces storage constraints by limiting the number of retained scans and processing outputs. These policies are intentionally separated from the signal processing pipeline, allowing them to be modified or extended without impacting the execution of core processing stages.

This design choice supports adaptability and maintainability, particularly in scenarios where storage constraints, naming conventions, or data management requirements may evolve over time.

7.3.5 Producer–Consumer Pattern

To support distributed processing and decoupling between system components, the Producer–Consumer pattern is employed. This pattern separates the generation of data from its consumption,

allowing each to operate independently and asynchronously.

In this implementation, the Raspberry Pi acts as the producer, generating FFT output files as part of the processing pipeline. These outputs are then transferred to an external operator workstation, which acts as the consumer. Data transfer is performed using `rsync`, enabling reliable and incremental synchronization of output files.

This separation allows the embedded system to focus on real-time data acquisition and processing, while more computationally intensive analysis and visualization are performed on the external system. The use of a file-based exchange mechanism further enhances robustness, as it does not require persistent network connections or tightly synchronized execution.

7.3.6 Layered Architecture

In addition to the primary design patterns described above, the system exhibits characteristics of a layered architectural structure. The system is organized into distinct functional layers, including audio capture, signal transformation, spatial processing, frequency analysis, and external data processing.

Each layer is responsible for a specific aspect of system functionality and communicates with adjacent layers through well-defined interfaces. This separation of concerns improves maintainability and enables targeted enhancements within individual layers. For example, improvements to beamforming algorithms or FFT processing can be implemented without requiring changes to the audio capture or data transfer layers.

This layered organization complements the pipeline structure and reinforces the system's modular design.

8 Results

The system successfully implemented the full audio processing pipeline, including multichannel audio capture, channel decomposition, and directional beamforming. The Raspberry Pi consistently captured short-duration multichannel recordings using the connected audio interface and stored them in the designated scan directory.

Channel decomposition using FFmpeg produced individual microphone recordings for each scan cycle. These files were verified to contain distinct audio signals corresponding to each input channel.

The beamforming stage produced directional audio outputs across multiple angles. For each scan, a set of beamformed waveform files was generated, corresponding to predefined directional increments. These outputs exhibited variation in amplitude and signal characteristics across angles, indicating that directional processing was functioning as intended.

The frequency-domain analysis stage was successfully invoked as part of the processing pipeline. However, due to file output handling constraints during execution, the resulting FFT text outputs were not retained for inclusion in this report. Execution of this stage was verified through runtime behavior and console output, confirming integration within the system.

Overall, the system demonstrated consistent end-to-end operation from audio capture through beamforming, with verified execution of the frequency analysis stage.

9 Opportunities for Expansion

This system provides the foundation for expansion into larger microphone systems, especially given a stronger computational center or FPGA processing of demanding processes such as beamforming and FFT analysis. Likewise, the use of an RF antennae capable of reading RF Frequencies can alert the system when it should be observing new audio input. Multiple RF antennae have been a popular choice for similar detection arrays, but it is a much more expensive and harder to access solution.

10 Conclusion

This project demonstrates the capability of small, affordable, and readily available components are capable of forming basic detection and alarm systems for applications such as drone defense. These components include the mathematical applications necessary to perform these analyses, making the need for mathematical or coding acumen minimal with the proper preparation. This follows the Ukrainian example of accessible drone defense within the reach of the public.

11 Works Cited

References

[1] Anderssén, Victor. *c_fft_visualizer*. GitHub, 2024, `github.com/Avicted/c_fft_visualizer`. Accessed 4 May 2026.

[2] "Factory Method Pattern." *GeeksforGeeks*, 19 Sept. 2023, `www.geeksforgeeks.org/system-design/factory-method-for-designing-pattern/`. Accessed 4 May 2026.

[3] Harris, C. R., et al. "Array Programming with NumPy." *Nature*, vol. 585, 2020, pp. 357–362.

[4] "Observer Pattern — Set 1 (Introduction)." *GeeksforGeeks*, 22 Jan. 2024, `www.geeksforgeeks.org/system-design/observer-pattern-set-1-introduction/`. Accessed 4 May 2026.

[5] Python Software Foundation. *Python 3*. Python Software Foundation, 2026, `docs.python.org/3/`. Accessed 4 May 2026.

[6] Virtanen, Pauli, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods*, vol. 17, 2020, pp. 261–272.

[7] Grabowski, Mariusz. "Developing a Real-Time SDR System." Semiconductor Engineering, 29 July 2021, `semiengineering.com/developing-a-real-time-sdr-system/`. Accessed 4 May 2026.

12 Appendix

12.1 User Interface

12.1.1 Graphs.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SigInt Project</title>
  <link rel="stylesheet" href="styles.css">

  <!-- Chart.js CDN -->
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

  <style>
    .chart-panel {
      background: transparent;
      height: 400px;
      width: 100%;
      max-width: 1400px;
      margin: 40px auto;
    }

    .chart-panel canvas {
      width: 100% !important;
      height: 100% !important;
    }

    /* Remove extra spacing */
    h1 {
      margin-bottom: 10px;
    }

    .hero-box .overlay > div {
      display: flex;
      flex-direction: column;
      gap: 10px;
    }
  </style>
</head>

<body>
  <main>
```

```
  <div class="hero-box">
    <div class="overlay">
      <div>
        <h1>Graphs</h1>

        <section id="sigint-options">
          <button onclick="window.location.href='index.html'">Home</button>
          <button onclick="window.location.href='graphs.html'">Graphs</button>
          <button onclick="window.location.href='radar.html'">Radar</button>
        </section>

        <!-- ONLY MIC CHARTS -->
        <div class="chart-panel">
          <h3>Mic 1</h3>
          <canvas id="chartMic1"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 2</h3>
          <canvas id="chartMic2"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 3</h3>
          <canvas id="chartMic3"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 4</h3>
          <canvas id="chartMic4"></canvas>
        </div>

      </div>
    </div>
  </div>
</main>

<script>
  // === MIC CHARTS ONLY ===
  const BIN_COUNT = 4097;
  const bins = Array.from({ length: BIN_COUNT }, (_, i) => i);

  const baseOptions = {
    type: 'line',
    data: {
      labels: bins.map(String),
```

```
    datasets: []
  },
  options: {
    responsive: true,
    maintainAspectRatio: false,
    interaction: { mode: 'nearest', intersect: false },
    plugins: {
      legend: { display: false }
    },
    scales: {
      x: { title: { display: true, text: 'Bin Number' } },
      y: { title: { display: true, text: 're' }, beginAtZero: false }
    }
  }
};

const chartMic1 = new Chart(
  document.getElementById('chartMic1').getContext('2d'),
  JSON.parse(JSON.stringify(baseOptions))
);

const chartMic2 = new Chart(
  document.getElementById('chartMic2').getContext('2d'),
  JSON.parse(JSON.stringify(baseOptions))
);

const chartMic3 = new Chart(
  document.getElementById('chartMic3').getContext('2d'),
  JSON.parse(JSON.stringify(baseOptions))
);

const chartMic4 = new Chart(
  document.getElementById('chartMic4').getContext('2d'),
  JSON.parse(JSON.stringify(baseOptions))
);

const API_URL = '/data/data.json';

async function fetchNewData() {
  try {
    const res = await fetch(API_URL);
    if (!res.ok) {
      console.error('API error', res.status);
      return;
    }
```

```
    const filesData = await res.json();

    const micMap = {
      1: filesData.find(f => /Mic_1/i.test(f.file)) || null,
      2: filesData.find(f => /Mic_2/i.test(f.file)) || null,
      3: filesData.find(f => /Mic_3/i.test(f.file)) || null,
      4: filesData.find(f => /Mic_4/i.test(f.file)) || null
    };

    function buildDataset(fileData, color) {
      if (!fileData) return [];

      const map = new Map();
      fileData.points.forEach(p => {
        map.set(Number(p.bin), Number(p.re));
      });

      const values = bins.map(bin => {
        const v = map.get(bin);
        return v !== undefined ? v : null;
      });

      return [{
        label: fileData.file,
        data: values,
        borderColor: color,
        backgroundColor: color.replace('1)', '0.2)'),
        borderWidth: 1,
        fill: false,
        spanGaps: true,
        pointRadius: 0,
        tension: 0
      }];
    }

    chartMic1.data.datasets = buildDataset(micMap[1], 'rgba(54, 162, 235, 1)');
    chartMic2.data.datasets = buildDataset(micMap[2], 'rgba(255, 99, 132, 1)');
    chartMic3.data.datasets = buildDataset(micMap[3], 'rgba(75, 192, 192, 1)');
    chartMic4.data.datasets = buildDataset(micMap[4], 'rgba(255, 206, 86, 1)');

    chartMic1.update();
    chartMic2.update();
    chartMic3.update();
    chartMic4.update();

  } catch (err) {
```

```
        console.error('Error fetching data:', err);
      }
    }

    fetchNewData();
    setInterval(fetchNewData, 5000);
  </script>
</body>
</html>
```

12.1.2 Logs.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SigInt Project - Logs</title>
  <link rel="stylesheet" href="styles.css">

  <style>
    #logs {
      margin-top: 20px;
      padding: 10px;
    }

    .log-entry {
      background: rgba(255,255,255,0.1);
      margin: 10px 0;
      padding: 10px;
      border-radius: 8px;
    }
  </style>
</head>

<body>
  <main>
    <div class="hero-box">
      <div class="overlay">
        <div>
          <h1>Logs</h1>

          <section id="sigint-options">
            <button onclick="window.location.href='index.html'">Home</button>
            <button onclick="window.location.href='radar.html'">Radar</button>
            <button onclick="window.location.href='graphs.html'">Graphs</button>
```

```
        </section>

        <!-- LOG DISPLAY -->
        <div id="logs"></div>

      </div>
    </div>
  </div>
</main>

<script>
  async function loadData() {
    try {
      const res = await fetch("data.json");
      const newData = await res.json();

      let history = JSON.parse(localStorage.getItem("scanHistory")) || [];

      const existingFiles = new Set(history.map(e => e.file));

      newData.forEach(entry => {
        if (!existingFiles.has(entry.file)) {
          entry.timestamp = Date.now();
          history.push(entry);
        }
      });
      localStorage.setItem("scanHistory", JSON.stringify(history));

      displayLogs(history);
    } catch (err) {
      console.error("Error loading data:", err);
    }
  }

  function displayLogs(history) {
    const container = document.getElementById("logs");

    // newest first
    const sorted = [...history].sort((a, b) => b.timestamp - a.timestamp);

    container.innerHTML = sorted.map(entry => `
      <div class="log-entry">
        <strong>${entry.file}</strong><br>
        Time: ${new Date(entry.timestamp).toLocaleString()}<br>
        Points: ${entry.points.length}
      </div>
```

```
      ').join("");
    }

    // refresh every second
    setInterval(loadData, 1000);
    loadData();
  </script>
</body>
</html>
```

12.1.3 Radar.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>SigInt Project - Radar</title>
  <link rel="stylesheet" href="styles.css" />

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

  <style>
    .chart-panel {
      background: transparent;
      height: 520px;
      width: 100%;
      max-width: 950px;
      margin: 40px auto;
    }

    .chart-panel canvas {
      width: 100% !important;
      height: 100% !important;
    }

    h1 {
      margin-bottom: 10px;
    }

    .hero-box .overlay > div {
      display: flex;
      flex-direction: column;
      gap: 10px;
      width: 100%;
      max-width: 1200px;
```

```
      margin: 0 auto;
    }

    #sigint-options {
      display: flex;
      flex-wrap: wrap;
      gap: 10px;
      justify-content: center;
      margin-bottom: 20px;
    }

    button {
      text-decoration: none;
      background-color: #555;
      color: #eee;
      padding: 8px 15px;
      border-radius: 5px;
      font-weight: bold;
      border: 3px solid #888;
      transition: all .2s linear;
      position: relative;
      top: 0px;
      display: inline-block;
      cursor: pointer;
    }

    button:hover {
      top: -3px;
      background-color: #333;
      color: #fff;
      border: 3px solid #555;
    }

    h3 {
      color: white;
      text-align: center;
      margin-bottom: 10px;
    }
  </style>
</head>

<body>
  <main>
    <div class="hero-box">
      <div class="overlay">
        <div>
```

```
        <h1>Radar Graphs</h1>

        <section id="sigint-options">
          <button onclick="window.location.href='index.html'">Home</button>
          <button onclick="window.location.href='graphs.html'">Graphs</button>
          <button onclick="window.location.href='radar.html'">Radar</button>
        </section>

        <div class="chart-panel">
          <h3>Mic 1</h3>
          <canvas id="chartMic1"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 2</h3>
          <canvas id="chartMic2"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 3</h3>
          <canvas id="chartMic3"></canvas>
        </div>

        <div class="chart-panel">
          <h3>Mic 4</h3>
          <canvas id="chartMic4"></canvas>
        </div>
      </div>
    </div>
  </div>
</main>

<script>
  const BIN_COUNT = 4097;

  // Smaller step = more spokes/labels around the circle
  const SAMPLE_STEP = 128;

  const sampledBins = Array.from(
    { length: Math.ceil(BIN_COUNT / SAMPLE_STEP) },
    (_, i) => i * SAMPLE_STEP
  ).filter(bin => bin < BIN_COUNT);

  function createRadarChart(canvasId) {
    return new Chart(document.getElementById(canvasId).getContext('2d'), {
      type: 'radar',
```

```
data: {
  labels: sampledBins.map(String),
  datasets: []
},
options: {
  responsive: true,
  maintainAspectRatio: false,
  animation: false,
  plugins: {
    legend: {
      display: false
    },
    tooltip: {
      enabled: true
    }
  },
  elements: {
    line: {
      borderWidth: 2,
      tension: 0
    },
    point: {
      radius: 1.5,
      hoverRadius: 4
    }
  },
  scales: {
    r: {
      beginAtZero: true,

      angleLines: {
        display: true,
        color: 'rgba(255,255,255,0.45)',
        lineWidth: 1.5
      },

      grid: {
        circular: true,
        color: 'rgba(255,255,255,0.28)',
        lineWidth: 1.2
      },

      pointLabels: {
        color: 'white',
        padding: 24,
        centerPointLabels: false,
```

```
            font: {
              size: 10
            }
          },

          ticks: {
            count: 6,
            color: 'white',
            backdropColor: 'transparent',
            showLabelBackdrop: false
          }
        }
      }
    }
  });
}

const chartMic1 = createRadarChart('chartMic1');
const chartMic2 = createRadarChart('chartMic2');
const chartMic3 = createRadarChart('chartMic3');
const chartMic4 = createRadarChart('chartMic4');

const API_URL = '/data/data.json';

async function fetchNewData() {
  try {
    const res = await fetch(API_URL);
    if (!res.ok) {
      console.error('API error', res.status);
      return;
    }

    const filesData = await res.json();

    const micMap = {
      1: filesData.find(f => /Mic_1/i.test(f.file)) || null,
      2: filesData.find(f => /Mic_2/i.test(f.file)) || null,
      3: filesData.find(f => /Mic_3/i.test(f.file)) || null,
      4: filesData.find(f => /Mic_4/i.test(f.file)) || null
    };

    function buildDataset(fileData, color) {
      if (!fileData) return [];

      const map = new Map();
      fileData.points.forEach(p => {
```

```
            map.set(Number(p.bin), Number(p.re));
          });

          const values = sampledBins.map(bin => {
            const v = map.get(bin);
            return v !== undefined ? v : null;
          });

          return [{
            label: fileData.file,
            data: values,
            borderColor: color,
            backgroundColor: color.replace('1)', '0.20)'),
            pointBackgroundColor: color,
            pointBorderColor: color,
            pointHoverBackgroundColor: '#ffffff',
            pointHoverBorderColor: color,
            fill: true,
            spanGaps: true
          }];
        }
        chartMic1.data.datasets = buildDataset(micMap[1], 'rgba(54, 162, 235, 1)');
        chartMic2.data.datasets = buildDataset(micMap[2], 'rgba(255, 99, 132, 1)');
        chartMic3.data.datasets = buildDataset(micMap[3], 'rgba(75, 192, 192, 1)');
        chartMic4.data.datasets = buildDataset(micMap[4], 'rgba(255, 206, 86, 1)');

        chartMic1.update();
        chartMic2.update();
        chartMic3.update();
        chartMic4.update();
      } catch (err) {
        console.error('Error fetching data:', err);
      }
    }

    fetchNewData();
    setInterval(fetchNewData, 5000);
  </script>
</body>
</html>
```

12.1.4 SigIntParser.lua

```
local lfs = require("lfs")

-- ===== CONFIG =====
```

```
local local_folder = "LOCALFILENAME HERE"   -- change
local remote_user = "team19"
local remote_host = "192.168.1.123"
local remote_dir  = "/home/team19/scan"

-- ===== helpers =====
local function run_rsync()
    {print("Running rsync...")
    local ok, reason, code = os.execute(rsync_cmd)
    {print("rsync result:", ok, reason, code)
end

local function read_file(path)
    local f = io.open(path, "r")
    if not f then
        return nil
    end
    local content = f:read("*all")
    f:close()
    return content
end

local function process_file(path)
    local content = read_file(path)
    if not content then
        {print("Could not read:", path)
        return
    end

    {print("Processing:", path)
    {print(content)

end

while true do
    run_rsync()

    for file in lfs.dir(localfolder) do
        if file:match("^Scan%d+Mic%d+_output%.txt$") then
            local full = local_folder .. "/" .. file
            if not seen[full] then
                seen[full] = true
                process_file(full)
            end
        end
    end
```

```
    -- sleep ~0.75 second
    if package.config:sub(1,1) == "\" then
        os.execute("ping -n 2 127.0.0.1 > nul")
    else
        os.execute("sleep 0.75")
    end
end
```

12.2 Main Processing Pipeline Script

```
[language=bash, caption={run\_mixpre\_fft\_loop.sh}]
#!/usr/bin/env bash
set -euo pipefail

DEVICE="plughw:3,0"
FORMAT="S24_3LE"
RATE="48000"
CHANNELS="4"

SCAN_DIR="/home/team19/scan"
BEAM_DIR="/home/team19/beamform_outputs"
FFT_DIR="/home/team19/fft_output"

VISUALIZER="/home/team19/Desktop/capstone/c_fft_visualizer/build/c_fft_visualizer"
BEAMFORMER="/home/team19/beamform.py"
PYTHON_BIN="/home/team19/beam_env/bin/python"

CAPTURE_LEN="0.25"
SLEEP_GAP="0.50"
KEEP_SETS=20

mkdir -p "$SCAN_DIR" "$BEAM_DIR" "$FFT_DIR"

cleanup_old_scan_sets() {
    mapfile -t scans < <(
        find "$SCAN_DIR" -maxdepth 1 -type f -name 'Scan_*.wav' \
        | sed -E 's|_Mic_[1-4]\.wav$||; s|\.wav$||' \
        | sort -u
    )
    local count="${#scans[@]}"
    if (( count <= KEEP_SETS )); then return; fi
    local remove_count=$((count - KEEP_SETS))
    for ((i=0; i<remove_count; i++)); do
        base="${scans[$i]}"
        rm -f "${base}.wav" "${base}_Mic_"*.wav
```

```
    done
}

echo "Starting pipeline..."

scan_num=1

while true; do

    scan_id=$(printf "%03d" "$scan_num")
    multi="${SCAN_DIR}/Scan_${scan_id}.wav"

    timeout --signal=INT "${CAPTURE_LEN}s" \
        arecord -D "$DEVICE" -f "$FORMAT" -r "$RATE" -c "$CHANNELS" "$multi" || true

    ffmpeg -y -i "$multi" -filter:a "pan=mono|c0=c0" "${SCAN_DIR}/Scan_${scan_id}_Mic_1.wav"
    ffmpeg -y -i "$multi" -filter:a "pan=mono|c0=c1" "${SCAN_DIR}/Scan_${scan_id}_Mic_2.wav"
    ffmpeg -y -i "$multi" -filter:a "pan=mono|c0=c2" "${SCAN_DIR}/Scan_${scan_id}_Mic_3.wav"
    ffmpeg -y -i "$multi" -filter:a "pan=mono|c0=c3" "${SCAN_DIR}/Scan_${scan_id}_Mic_4.wav"

    "$PYTHON_BIN" "$BEAMFORMER" "$scan_id" "$SCAN_DIR" "$BEAM_DIR"

    for beam_wav in "$BEAM_DIR"/Scan_${scan_id}_Angle_*.wav; do
        out_txt="$FFT_DIR/$(basename "${beam_wav%.wav}.txt")"
        xvfb-run -a "$VISUALIZER" "$beam_wav" > "$out_txt" 2>&1 || true
    done

    cleanup_old_scan_sets

    ((scan_num++))
    sleep "$SLEEP_GAP"

done
```

12.3 Beamforming Script (Python)

```
[language=Python, caption={beamform.py}]
import numpy as np
import scipy.io.wavfile as wav
import os
import sys

# ============================================================
# CONFIGURATION
# ============================================================
```

```
NUM_MICS = 4
SPEED_OF_SOUND = 343.0  # meters per second
MIC_SPACING = 0.05      # meters (adjust to physical array)

ANGLES = range(0, 360, 30)  # 12 directions

# ============================================================
# LOAD INPUT MICROPHONE SIGNALS
# ============================================================

def load_mics(scan_id, input_dir):
    signals = []
    sample_rate = None

    for i in range(1, NUM_MICS + 1):
        path = os.path.join(input_dir, f"Scan_{scan_id}_Mic_{i}.wav")
        sr, data = wav.read(path)

        if sample_rate is None:
            sample_rate = sr

        signals.append(data.astype(np.float32))

    return sample_rate, signals

# ============================================================
# COMPUTE DELAYS FOR A GIVEN ANGLE
# ============================================================

def compute_delays(angle_deg, sample_rate):
    angle_rad = np.deg2rad(angle_deg)

    delays = []
    for mic_idx in range(NUM_MICS):
        distance = mic_idx * MIC_SPACING * np.sin(angle_rad)
        time_delay = distance / SPEED_OF_SOUND
        sample_delay = int(time_delay * sample_rate)
        delays.append(sample_delay)

    return delays

# ============================================================
# APPLY DELAYS AND SUM SIGNALS (BEAMFORMING)
# ============================================================

def apply_beamforming(signals, delays):
```

```
    aligned = []

    for sig, delay in zip(signals, delays):
        if delay > 0:
            padded = np.pad(sig, (delay, 0))[:len(sig)]
        elif delay < 0:
            padded = sig[-delay:]
            padded = np.pad(padded, (0, len(sig) - len(padded)))
        else:
            padded = sig

        aligned.append(padded)

    beam = np.sum(aligned, axis=0)
    beam /= len(aligned)

    return beam.astype(np.int16)

# ============================================================
# MAIN BEAMFORMING FUNCTION
# ============================================================

def beamform(scan_id, input_dir, output_dir):
    os.makedirs(output_dir, exist_ok=True)

    sample_rate, signals = load_mics(scan_id, input_dir)

    for angle in ANGLES:
        delays = compute_delays(angle, sample_rate)
        beam = apply_beamforming(signals, delays)

        out_path = os.path.join(
            output_dir,
            f"Scan_{scan_id}_Angle_{angle}.wav"
        )

        wav.write(out_path, sample_rate, beam)

        print(f"Generated {out_path}")

# ============================================================
# ENTRY POINT
# ============================================================

if __name__ == "__main__":
    if len(sys.argv) != 4:
```

```
        print("Usage: beamform.py <scan_id> <input_dir> <output_dir>")
        sys.exit(1)

    scan_id = sys.argv[1]
    input_dir = sys.argv[2]
    output_dir = sys.argv[3]

    beamform(scan_id, input_dir, output_dir)
```

12.4 Github

The link to the remainder of the code is: `https://github.com/MJParker19/483SigInt`