Radio Diversity Automation

Group 26:
Kim Hoang
John Crawford
Jurgis Siusys

Meet the Team

John Crawford

Electrical Engineering

Jurgis Siusys

Computer Engineering

Kim Hoang

Computer Engineering

Introduction

- Original project: passive aircraft detection system with sponsor funding
- Funding was pulled → project pivoted to new scope
- Current focus: **radio diversity system** using one broadcast and three receivers into a single SDR
- DSP algorithms select the strongest, most reliable link in real time
- Demonstrates reliability with low-cost COTS parts, scalable for future localization

Motivation & Background

- Reliable wireless links are critical in environments with fading, noise, and interference
- Single-antenna receivers often lose signal quality due to multipath and obstructions
- Radio diversity improves reliability by comparing multiple antenna branches
- Using low-cost COTS parts makes this approach affordable and easy to replicate
- This proof-of-concept can be extended to larger systems, including future localization

Goals & Objectives

Goals

- Design a 3-antenna diversity receiver using a single SDR platform
- Demonstrate real-time link selection with measurable improvement over single-antenna baseline
- Keep cost low with Amazon-sourced COTS hardware

Objectives

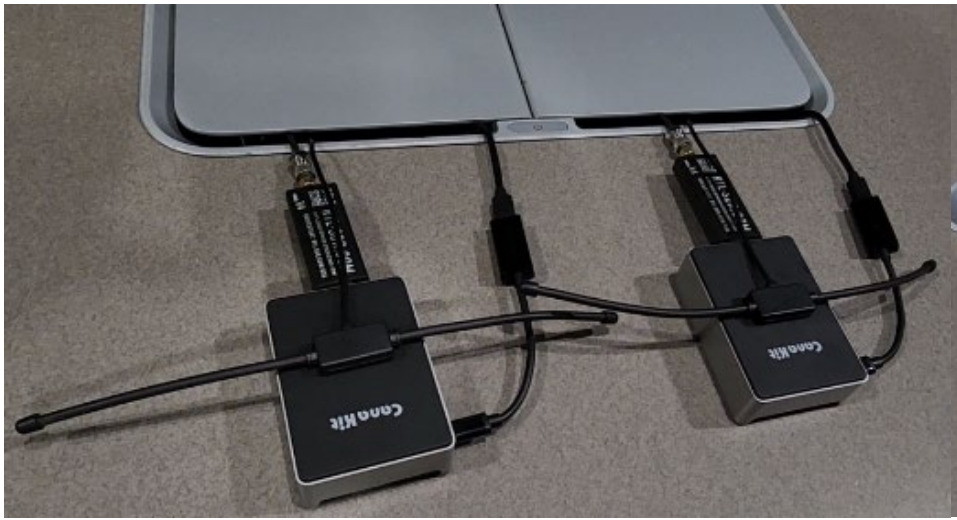
- Implement DSP algorithms to calculate SNR/RSSI per branch
- Apply hysteresis/smoothing to avoid unstable switching
- Achieve selection latency < 50 ms (parallel) / < 150 ms (multiplexed)
- Validate improvement through controlled lab/field tests

Engineering Specifications

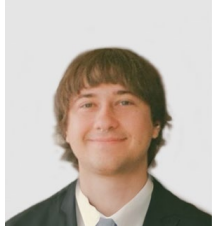
Parameter	***Target Value***	***Notes***
Total Cost	< $400	Amazon COTS Components
Receiver Branches	3 Antennas / Receivers	Minimum required for diversity benefit
Operating Band	FM 88-108MHz	Legal and accessible signals for proof of concept
Fading Signal and Switching Latency	< 1s to detect fading signal and < 50ms to switch	Quick switching between focused SDRs
Audio Delay	< 3s	Ensures smooth switching of audio between SDRs
Diversity Gain	> 6dB SNR improvement	Demonstrates benefit of diversity system
Power Requirement	< 10W	Can run on USB bank or wall plug
Touchscreen Latency	< 500ms	Time between button press and touchscreen response
Selection Accuracy	90%	Accuracy of highest SNR being selected at any given moment.

Product Design

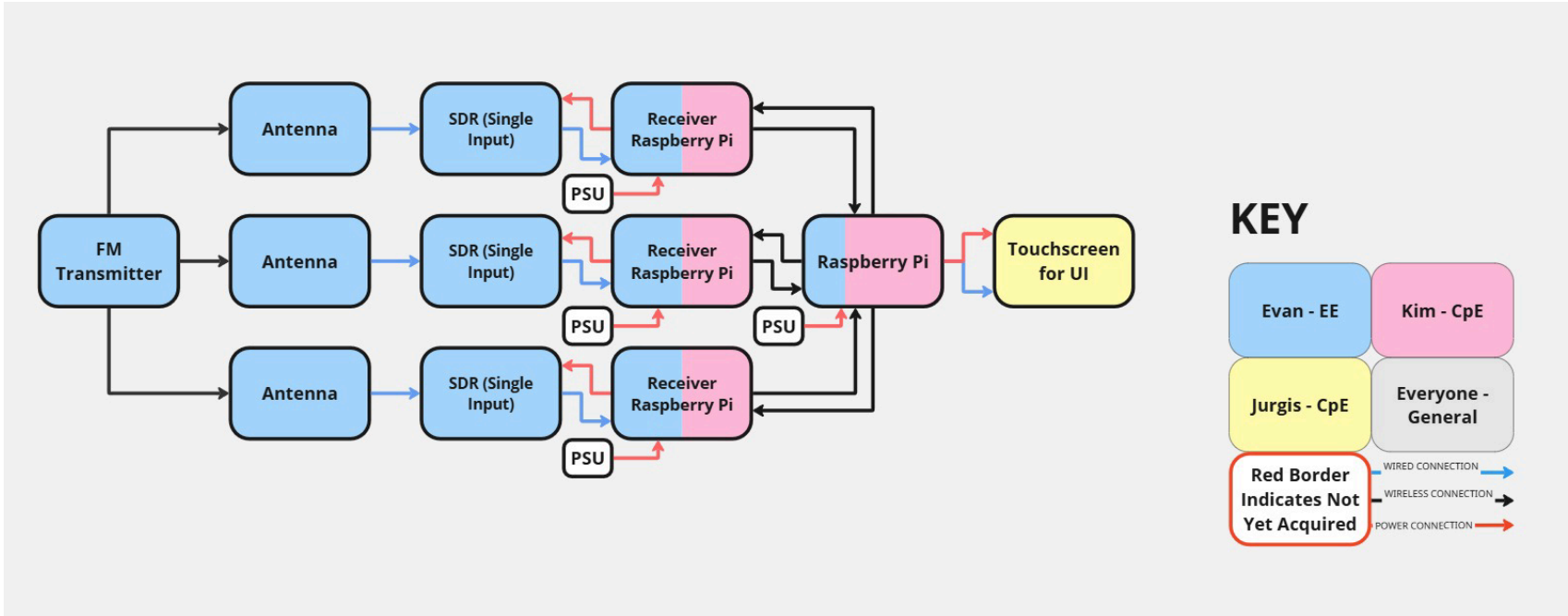
Receiver Pi Units



Central Pi Unit



Hardware Block Diagram



Hardware: MCU Selection

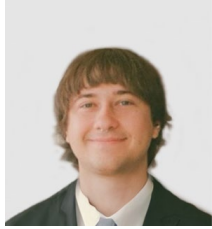
Processing Power:
- Quad-core ARM Cortex-A53 processor @ 1.2 GHz
- Sufficient for RF signal processing, data handling, and diversity testing

Integrated Design:
- Built-in storage, networking, and GPIO
- Software Compatibility:
  - Runs Linux-based OS
  - Access to machine learning frameworks & DSP tools

Connectivity & Expandability:
- USB 2.0 ports for SDR connections
- GPIO/SPI for touchscreen integration

Lower cost than newer models, strong community support, energy-efficient, and sufficient for near-range radio diversity experiments

Feature	***MCU ((Microcontroller Unit)***	***SBC (Single Board Computer***
Integration	Includes CPU, RAM, Flash storage, I/O on a single chip.	A complete computing system on a single board.
Processing Power	Low to Medium (10-800MHz)	High (Above 1 GHz).
Operating System	No OS or runs RTOS like FreeRTOS	Supports full OS like Linux, Android, Windows.
Programming Languages	Primarily C, C++, Assembly, and MicroPython	Supports Python, Java, C++, Go, Linux-based tools
Communication Protocols	Low level like I²C, SPI, UART	Supports high-speed data transfer like Wi-Fi, USB.
Peripheral Control	Optimized for direct hardware control (e.g., sensors, motors, displays)	Handles complex peripherals like cameras, touchscreens, GPUs, & interfaces.



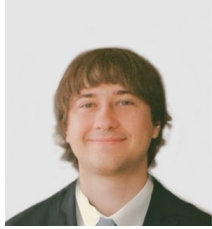
Hardware: SDR Selection

We selected 3 × RTL-SDR v4 units connected via a powered USB hub.

- Affordability, flexibility, and ease of replacement with tight budget constraints.
- We can test frequency, time, and spatial diversity.
- Provides a clear upgrade path: higher-end, multi-input SDRs could be adopted later if funding improves.
- Reflects the project's pivot from professional-grade equipment towards a cost-conscious proof of concept.

Feature	Multiple Single-Input SDRs (RTL-SDR v4)	Multi-Input SDR (e.g., LimeSDR Mini 2.0)
Cost	$30 each (3 units ≈ $90 total)	$300–400+ per device
Synchronization	Independent clocks	Shared clocking, phase-coherent inputs
Scalability	Easy to add/replace units	Fixed channel count
Ease of Use	Well-documented, strong community support	More complex drivers & setup
Performance	Good for proof-of-concept diversity	Supports advanced methods (TDOA, beamforming)

Hardware: Antenna Selection



3 Dipole Antenna - 1 for each channel

- **Cost-effective –** simple design, affordable, easy to replace
- **Reliable performance –** predictable omnidirectional radiation pattern
- **Easy integration –** straightforward to construct and tune for project frequencies
- **Compatibility –** well-matched with RTL-SDRs and Raspberry Pi testbed
- **Practical choice –** balances performance and budget for diversity experiments

Antenna	Frequency	Gain	Directional	Size	Pros	Cons
LPDA	Wide	High	Yes	Larger	Wide bandwidth, high gain	Larger complex positioning
Yagi-Uda	Narrow	High	Yes	Med	High gain, long-range detection	Requires precise alignment
Dipole	Med	Med	No	Small	Simple, low cost, omnidirectional	Lower range and gain
Monopole	Med	Low	No	Small	Low cost, omnidirectional	Lower gain and range

Hardware: Touchscreen Selection

Key Selection Criteria

- Balance of cost, responsiveness, size, and compatibility
- Must support real-time SDR monitoring and control
- Considered DSI (native), HDMI+USB, and third-party options
- Durability, brightness, and power consumption factored in

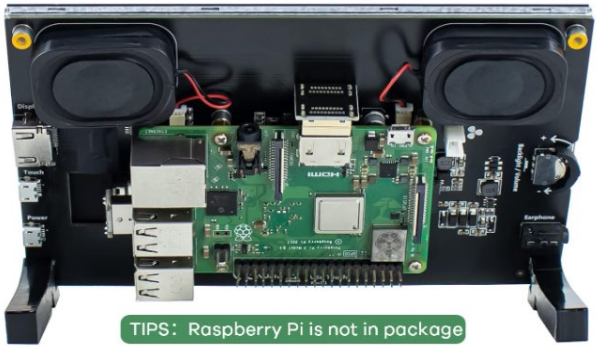
Final Selection

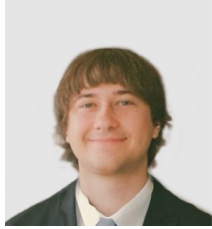
- **7” Raspberry Pi-Compatible Touchscreen**
  - 800×480 resolution provides a larger, clearer UI display
  - Connects using HDMI for video and USB for touch input
  - Bigger screen makes real-time metrics easier to read during testing
  - Suitable for extended GUI use and snapshot comparison features
- GUI Implementation
  - Python, optimized for small screens
  - Supports manual overrides & diagnostics
- Raspberry Pi 4 Model B is sufficient for GUI + detection services

Feature	7″ Touchscreen	3.5″ Touchscreen
Resolution	800×480 to 1024×600	480×320
Interface	DSI or HDMI + USB	GPIO / SPI
Power	~2–3W	~0.1–0.5W
Screen	Larger, easy to read	Compact, portable
Ports	HDMI + USB	GPIO/SPI
Cost	$50–$60	~$15–$25
Integration	More wiring, bigger	Easy housing, compact
Best For	Detailed plots & multi-window GUIs, extended GUI use	Logs, metrics, simple GUI, & portable, power-limited setups

Hardware: Touchscreen Audio & Input Integration

- Screen includes small built-in stereo speakers
- Audio output is handled through the screen’s HDMI connection
- Speakers are used for snapshot audio playback during testing
- Volume is sufficient for close-range field use
- Touchscreen input and audio both run through USB + HDMI





Hardware: Power Distribution Table

Central Pi Subsystem

Component	Voltage	Current (avg)	Current (peak)	Power (avg)
Raspberry Pi 4	5V	0.9A	1.6A	4.5W
7” Touchscreen	5V	1.1A	1.5A	5.5W
Total		**2A**	**3.1A**	**10W**

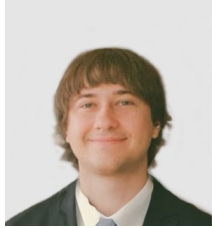
Receiver Pi Subsystem

Component	Voltage	Current (avg)	Current (peak)	Power (avg)
Raspberry Pi 4	5V	0.8A	1.5A	4.0W
RTL-SDR	5V	0.35A	0.5A	1.75W
Total		1.15A	2.0A	5.75W

Full System

Component	Voltage	Current (avg)	Current (peak)	Power (avg)
Raspberry Pi 4 (Central)	5V	.9A	1.6A	4.5W
7” Touchscreen	5V	1.1	1.5A	5.5W
Raspberry Pi 4 (Receiver) x 3	5V	2.7A	4.8A	13.5W
RTL-SDR x 3	5V	1.05A	1.5A	5.25W
Total		**5.75A**	**9.4A**	**28.75W**

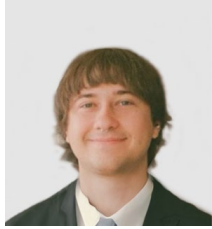
Hardware: Power Supply



Canakit USB-C Pi Power Supply

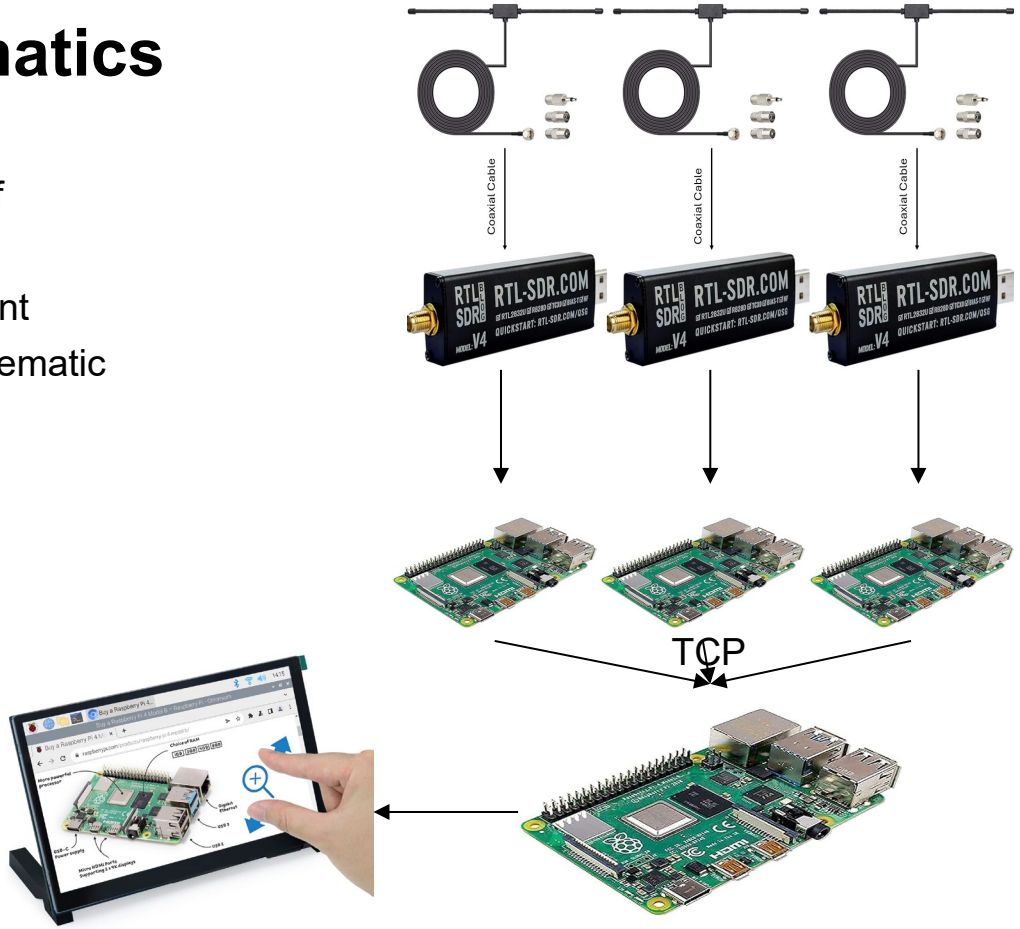
- Zero cost (already provided by Dr. Chan)
- Stable 5.1V regulated supply required for Pi 4 + peripherals
- Consistent performance under high CPU/GPU load
- Inline switch improves usability and reduces wear on USB-C connector
- Removes risk of low-quality 3rd-party adapters causing undervoltage errors

PSU	Pros	Cons	Suitability
Canakit USB-C Pi PSU	Designed for Raspberry Pi 4 (5.1V, 3.5A) & low ripple. Already available (no cost)	Requires AC wall power (not portable). Single unit power only	Most cost-effective and reliable, as it was provided to us
USB Battery Bank	Portability & easy field deployment	May not support sustained 3A load & runtime varies	Good as backup, not primary PSU
High Current 12V to 5V DC Buck Converter	Can run off solar or 12V battery & powers multiple Pis from one source	Must custom-wire power distribution. Requires additional fusing/protection	Best for permanent solar install, more complexity
Generic USB-C Wall Adapter	Cheap and widely available	Often limited to 5V 2A with high ripple voltage (can crash Pi)	Not recommended for Pi 4 or SDR loads

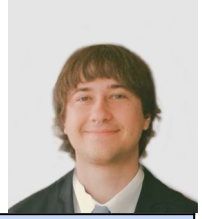


Hardware: Schematics

- Readily available, off the shelf products
- No significant PCB requirement
- Shown below is the block schematic for the whole system:



Communications: Message Transmission Methods

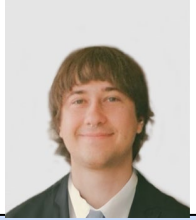


Off-the-shelf handheld FM Transmitters

- Extremely low cost compared to SDR transmitters
- Simple setup – no coding or custom modules required
- Provides a stable analog RF signal that SDRs can easily receive
- Supports variable distances, frequencies, and power levels for flexible testing
- Readily available, portable, and widely used, making it reliable for both lab and field tests

Method	Hardware	Pros	Cons
Zigbee	Zigbee radio module	Low power, mesh support	Very low data rate, short range
Z-Wave	Z-Wave module	Low power, interference resistant	Low bandwidth, short range
Bluetooth	USB dongle	Cheap, supported everywhere	~10 m range, interference
Wi-Fi	USB Wi-Fi adapter	High bandwidth, easy to set up	Power hungry, low outdoor range
LoRa	LoRa transceiver	Long range, very low power	Very low data rate
Analog FM (Transmitter)	Off-the-shelf handheld radio	Readily available, long range voice-band signal	No digital data, limited control

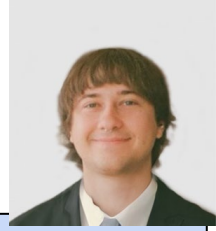
Communications: Wireless Methods



Wireless TCP connection over mobile hotspot

- Reliability for audio and JSON
  - Missing/mangled audio breaks the whole system
- No custom protocols needed
- Ease of use
- Completely able to run on phone hotspot

Method	Pros	Cons
Zigbee	Low power, mesh support, good for JSON	Requires extra hardware, too slow
Bluetooth	Low power and simple, doesn't need Wi-Fi	Very low bandwidth, latency spikes, poor for JSON
Wi-Fi: UDP	Low latency & simple, great for real time audio	No guaranteed delivery, packets can be lost or out of order
Wi-Fi: TCP	Guarantees clean transmission arrives in order	Higher latency, more overhead
LoRa	Best bandwidth for audio and supports TCP/UDP	Susceptible to Wi-Fi interference



Communications: Wired Methods

Hardware to Hardware	Method	Adapters	Notes
Antenna to SDR	Coaxial Cable	F type to SMA adapter	Slight signal loss from 75 ohm antenna to 50 ohm SDR, about 4dB
SDR to Raspberry Pi	USB	-----	USB also powers SDR
Raspberry Pi to Touchscreen	HDMI & USB	-----	HDMI for visuals, while USB powers screen and delivers touch capabilities

Software: Language Selection

Python vs Other High-Level Languages

Feature / Capability	Python	MATLAB	Node.js	Java
Real-Time Signal Processing (FM demod, filters, FFT)	Excellent — NumPy/SciPy, pyrtlsdr, fast enough for real-time	Great DSP tools but not real-time without Simulink	Not suitable for DSP	Moderate, but too heavy for real-time SDR
Hardware / SDR Integration	Native RTL-SDR support, GPIO/I2C/SPI on Raspberry Pi	Poor hardware control	Weak hardware access	Possible but complicated
Wireless Streaming (TCP/UDP)	Simple & reliable; ideal for SDR → central Pi streaming	Difficult	Good for web APIs, not IQ/audio streaming	Capable but verbose
Multithreading / Concurrency	Perfect for audio pipeline, jitter buffers, diversity logic	Limited	Strong async but not for DSP workloads	Strong but heavy to implement
GUI + Control Interface	Tkinter/PyQt — great for live SDR telemetry display	Decent (App Designer)	Must build as a web-app (overkill)	JavaFX works but heavy and slow

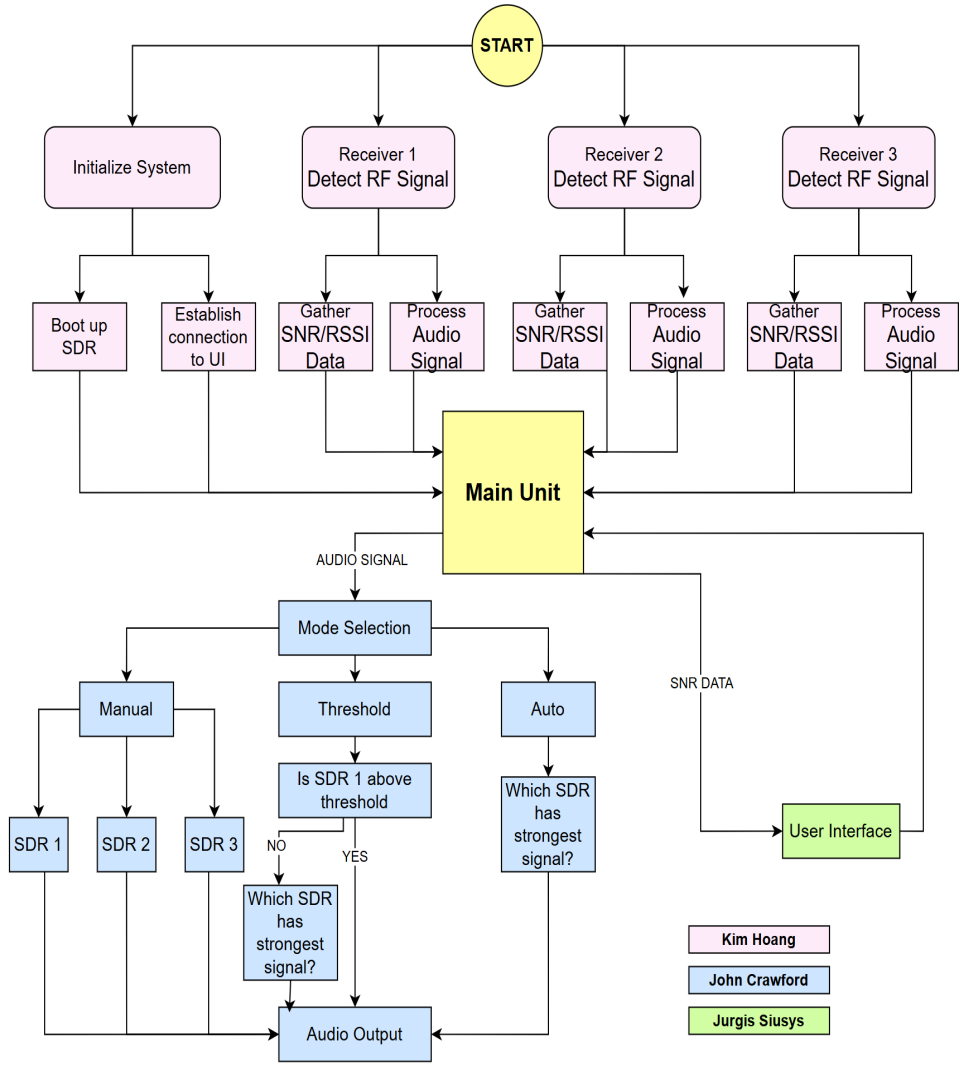
Software: Dev Environment

Python Development Environment Comparison

Environment / Tool	Strengths	Limitations
Thonny	Pre-installed on Raspberry Pi, easy to use, beginner-friendly.	Limited advanced features and Git integration.
PyCharm	Powerful IDE with great debugging and project tools.	Heavy on resources, slower on Pi.
Jupyter Notebook	Great for visualization and quick testing.	Poor for multi-file or structured projects.
Raspberry Pi OS (with IDLE)	Lightweight, built-in editor.	Basic, lacks advanced debugging or project tools.
Anaconda (Spyder)	Rich scientific computing support.	Bulky install, not optimized for embedded use.

Software Block Diagram

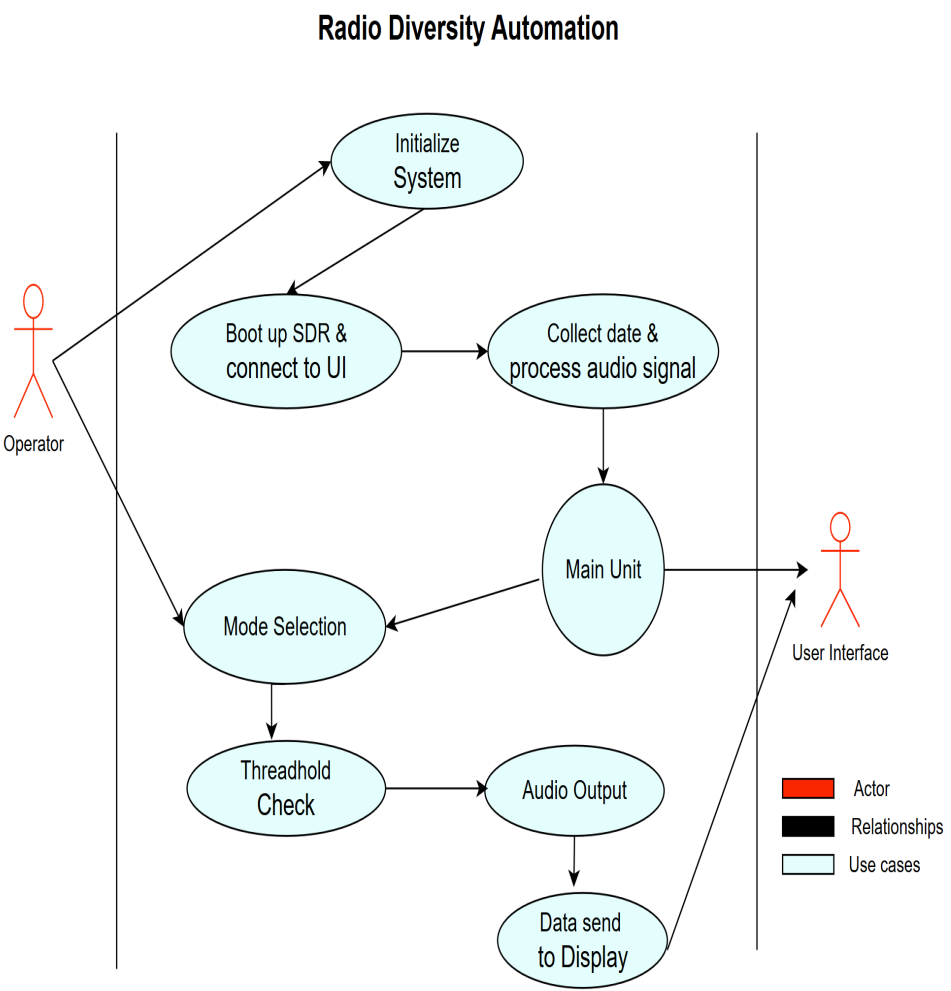
- Three SDR receivers detect RF signals and send audio + SNR/RSSI data to the Main Unit
- Main Unit compares signal quality and selects the best receiver
- Supports Manual, Threshold, and Auto mode selection
- Threshold mode checks SDR1's SNR before switching to strongest signal
- Auto mode automatically picks the highest-quality signal
- Selected SDR's audio is routed to output in real time
- UI displays live SNR, RSSI, and mode status for user control
- System initializes by booting SDRs and establishing wireless connection



Software Design

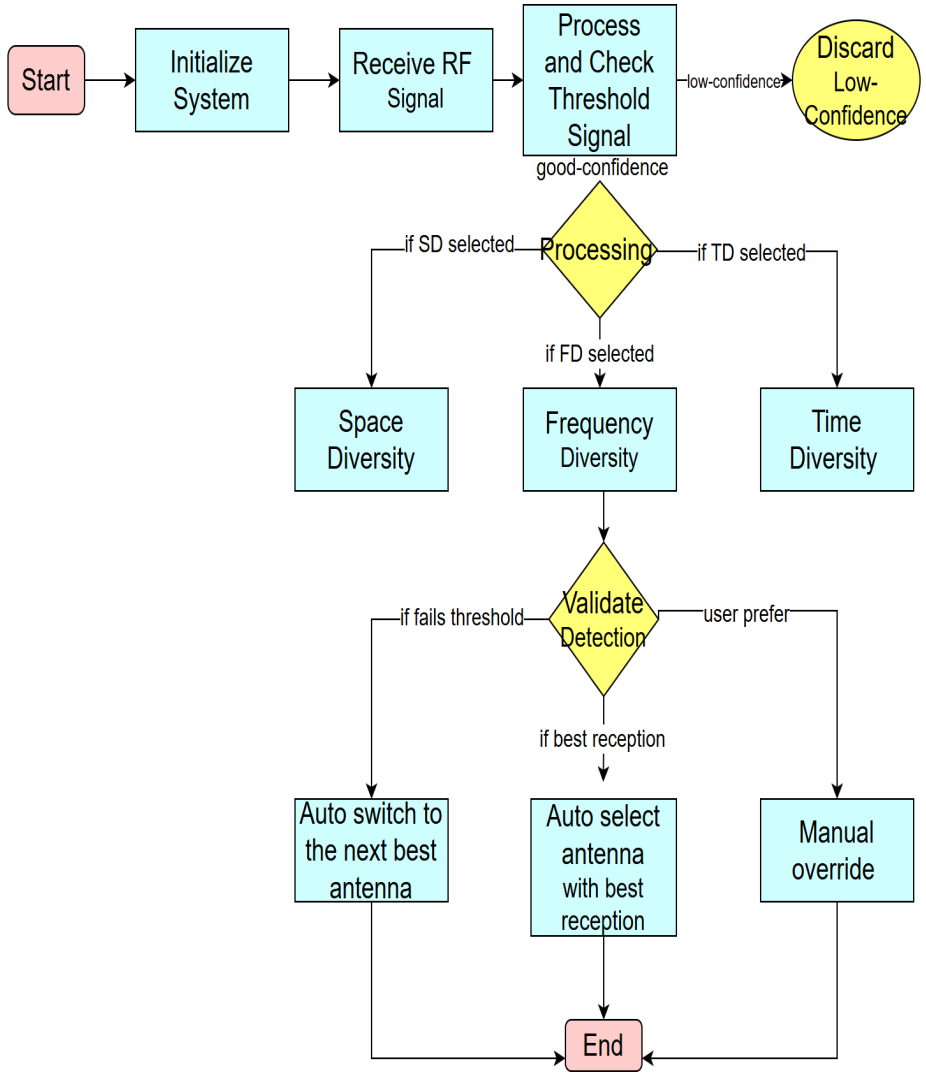
Use Case Diagram

- Operator initializes the system and boots the SDR receivers
- Receivers collect RF data and process audio signals
- Main Unit receives processed data and handles signal evaluation
- Operator selects operating mode (Manual, Threshold, Auto)
- System performs threshold checks and determines the best SDR
- Main Unit outputs selected audio to the user
- User Interface displays SNR, RSSI, and audio status to user



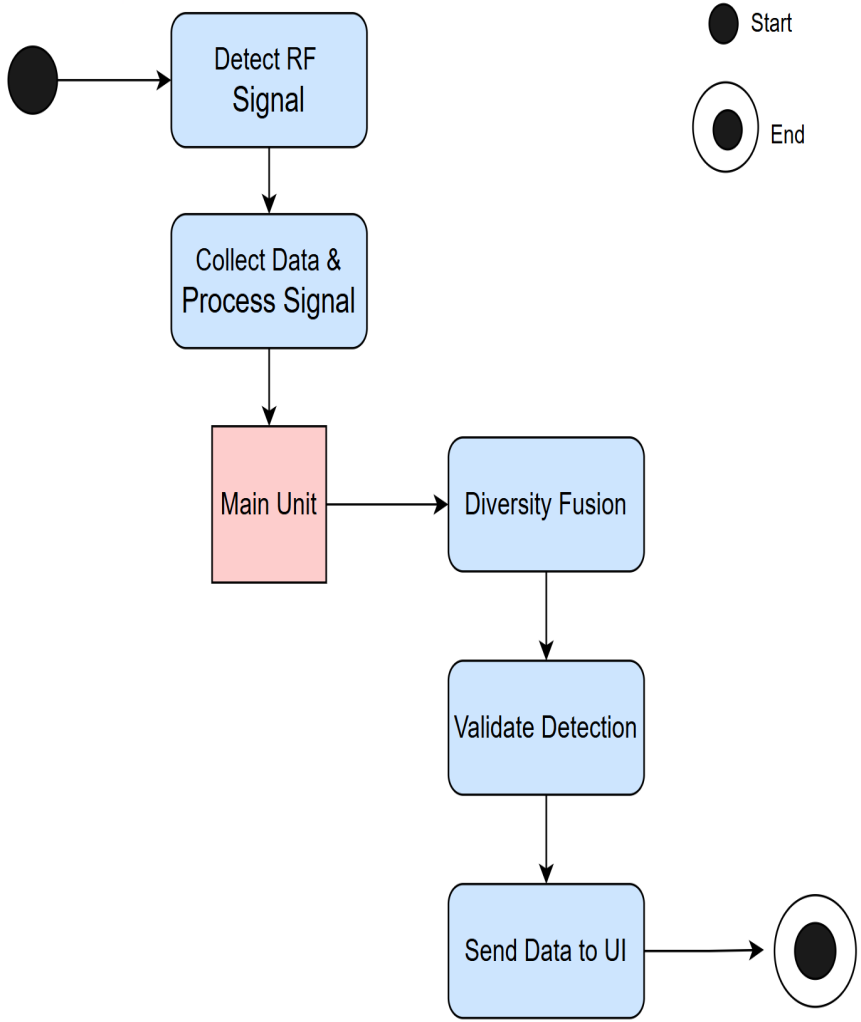
Software Design:
Subdiagram

- System initializes and begins receiving RF signals
- Incoming signals are checked against confidence thresholds
- Low-confidence signals are discarded
- Operator selects Space, Frequency, or Time Diversity mode
- Selected diversity method processes the signal
- Detection is validated using SNR/RSSI criteria
- If thresholds fail → system switches to next-best antenna
- If valid → system auto-selects antenna with best reception
- Manual override allows operator control at any time



Software Design:
Stage Diagram

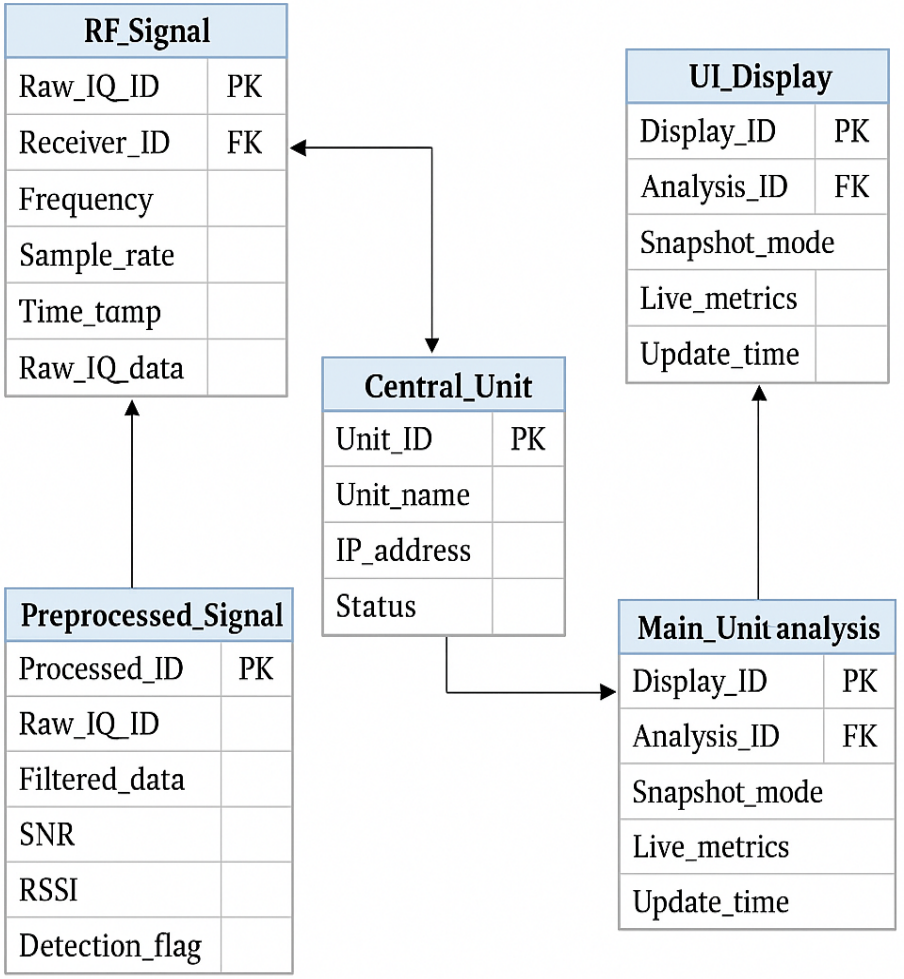
- Receiver Pi configures RTL-SDR and streams data via TCP.
- Central backend receives telemetry and audio in parallel threads.
- Backend buffers and selects audio based on mode logic.
- UI receives live metrics and sends user commands.



Software Design:

Entity-Relationship Diagram (ERD

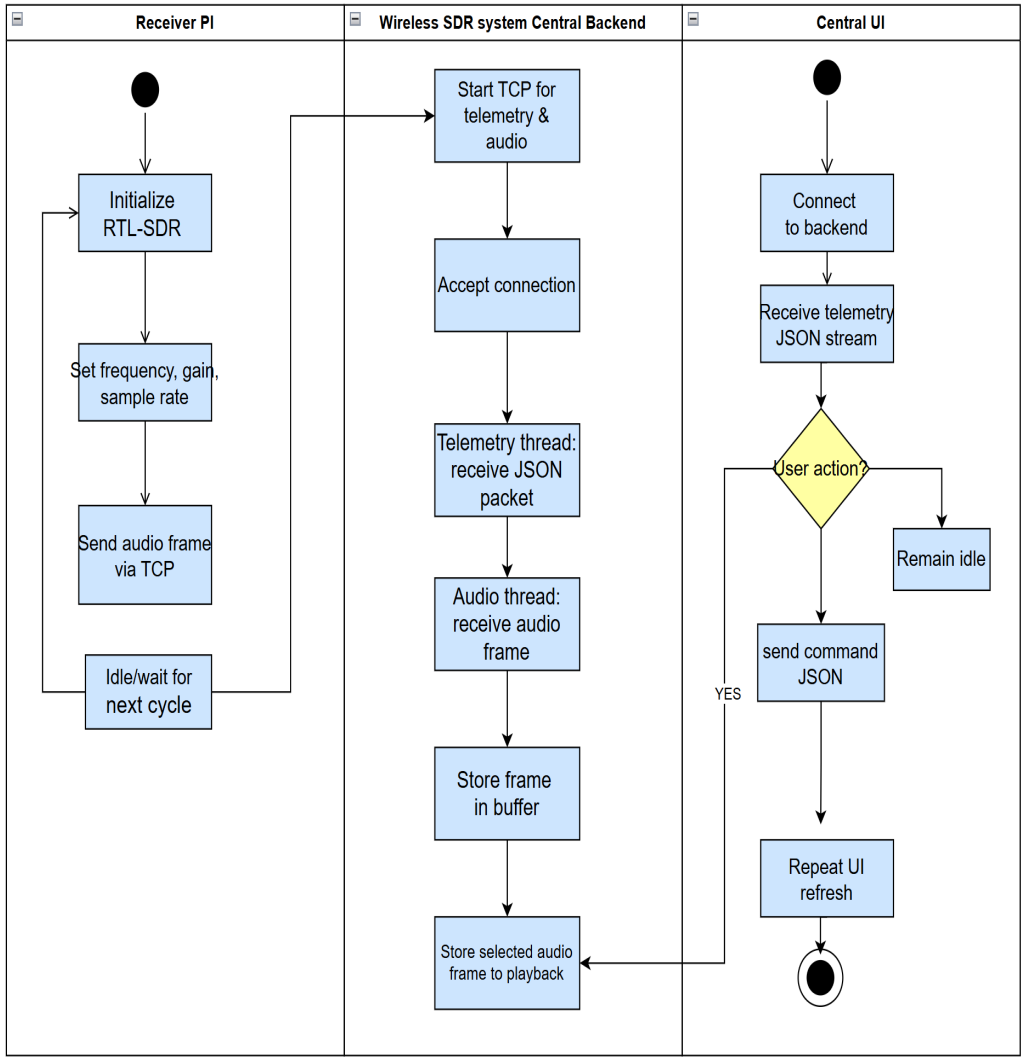
- Raw IQ data is stored in RF_Signal.
- Processed metrics (SNR/RSSI) saved in Preprocessed_Signal.
- Central_Unit tracks active system nodes.
- Main_Unit_Analysis holds diversity decisions.
- UI_Display logs what is shown to users.



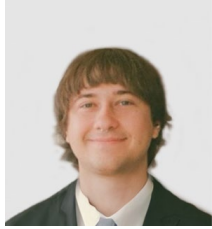
Software Design:

Activity Diagram

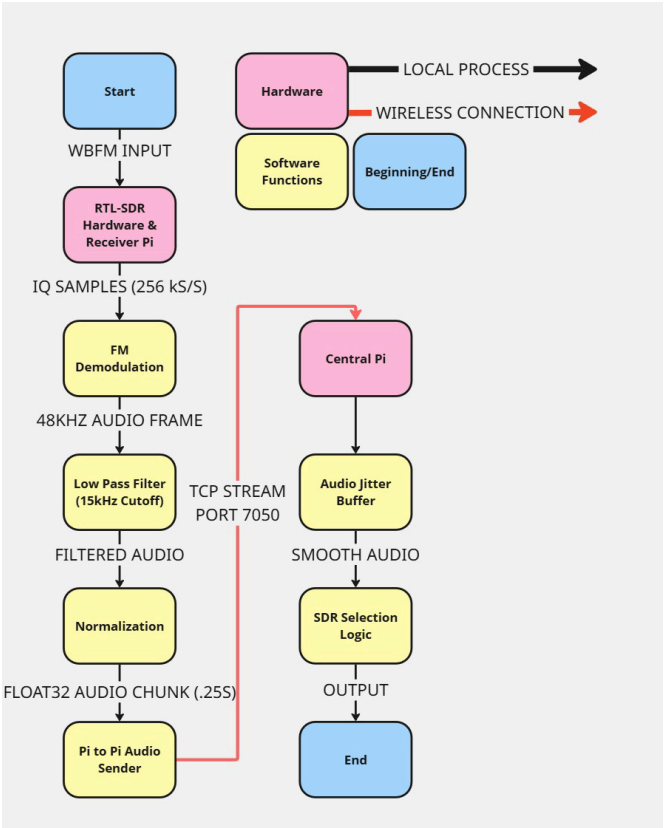
- Detect incoming RF signal.
- Extract metrics and preprocess data.
- Perform diversity fusion logic.
- Validate detection quality.
- Send output to UI.



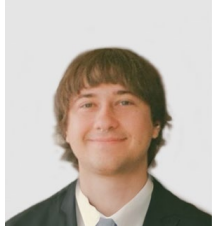
Software Design: Audio Streaming



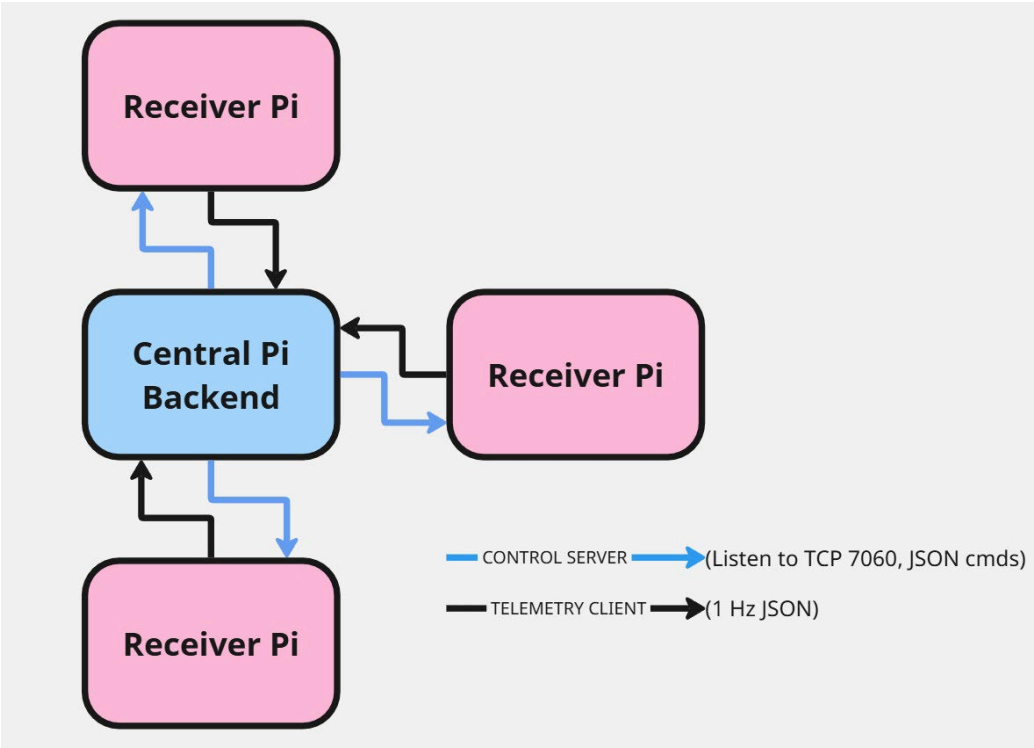
- RTL-SDR captures raw IQ at 256 kS/s
  - Samples are read continuously in chunks (CHUNK_SIZE = 32768) to avoid buffer underflow
  - IQ samples accumulate into a rolling buffer until a 0.25-second frame worth of samples is available
  - Pi performs FM demodulation, downsampling, and audio filtering.
  - Audio is normalized and transmitted as 0.25s float32 chunks.
  - Each chunk includes a sequence number for jitter-buffer reconstruction.
- Central Pi maintains per-SDR buffers and selects the correct SDR's stream.
- Final audio is written out in real time using sounddevice.
- System supports snapshot recording and muted playback of history.



Software Design: Wireless TCP Connection



- Every receiver PI connects to the central Pi via two persistent TCP sockets:
  - Telemetry → port 6050
  - Audio → port 7050
- Telemetry Info
  - RSSI
  - SNR
  - Frequency
  - Connected state
- Central Pi connects to the receiver Pi over a third TCP socket:
  - Control Channel → port 7060
  - Control channel updates RTL-SDR center frequency live
  - Control messages are simple JSON lines: {"cmd":"set_freq","value":98.1e6}
- Central Pi auto-reconnects to control servers and marks SDRs disconnected after timeout.



Why We Use TCP Instead of UDP

- Ensures ordered delivery of IQ/audio packets
- Provides guaranteed delivery (no silent packet loss)
- Built-in retransmission improves audio continuity
- Eliminates jitter from out-of-order packets, which caused choppy FM demodulation
- Better suited for large continuous data streams like raw IQ
- Simplifies buffering logic — no need to handle lost/partial packets manually
- More stable on Raspberry Pi wireless networks (hotspot/Wi-Fi)
- Reduces noise and artifacts caused by missing DSP sample frames

Software Design

User Interface (UI) Design

UI Component	Description	Implementation
SDR Status Cards	Show each SDR's connection, SNR, and tuned frequency. The user tap a card to select that SDR and switch between the two preset frequencies.	Built using Tkinter frames and labels. The cards update every second based on data received from the backend over a TCP socket.
Live Focus Panel	Display live signal information (SNR, RSSI, frequency) for whatever SDR is currently selected. Helps the user quickly see which receiver is performing best.	Tkinter panel that updates continuously using the shared state dictionary filled by the backend listener thread.
Snapshot & Comparison Panel	Saves up to two snapshots with timestamps and signal stats. A user can tap a snapshot to replay its audio and compare performance.	Dynamic Tkinter frames created when a snapshot is saved. Playback and saving commands are sent to the backend through JSON messages.
Mode & Playback Controls	Lets the user switch between manuel, Threshold, and Auto modes. Also includes buttons to take snapshots and play back the best snapshot.	Tkinter buttons that sent JSON commands to the backend to change modes, take snapshots, or trigger audio playback.

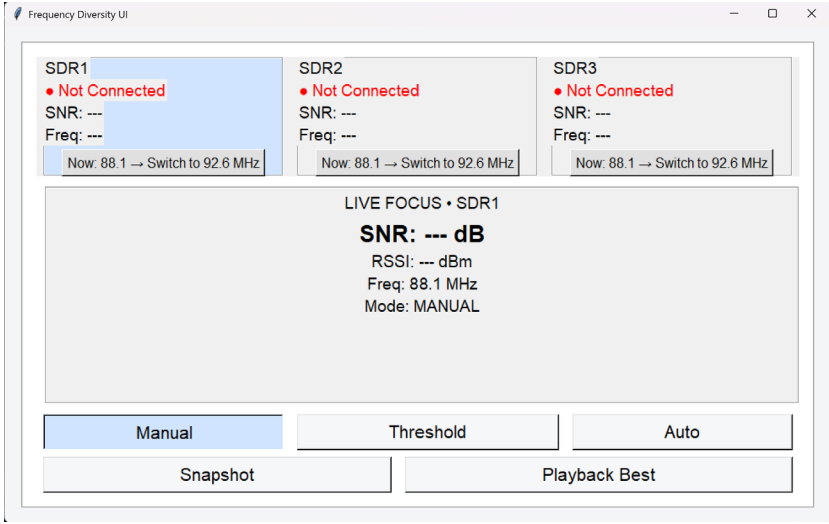
UI Overview

- Displays live data from all three SDR receivers
- Simple layout designed for fast decision-making

- Shows key stats: SNR, RSSI, and tuned frequency
- Built using lightweight Python Tkinter
- Continuously updated from backend data stream

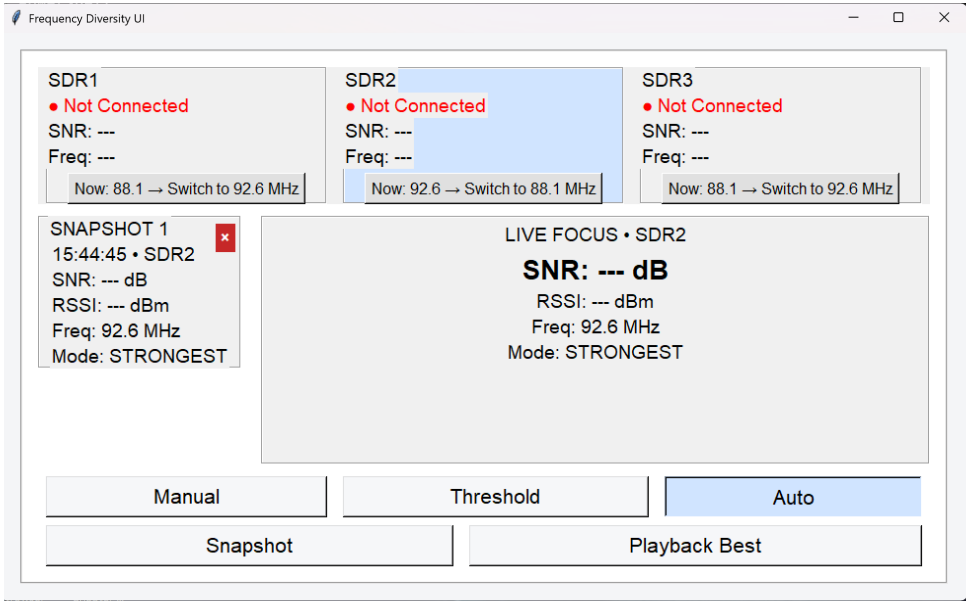
Purpose

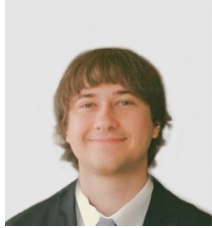
- Gives operators a clear, real-time picture of signal performance
- Makes it easy to see which SDR is strongest at any moment



UI Functionality & Features

- SDR status cards show connection and signal quality
- Tap to select an SDR and enter focused view
- Manual, Threshold, and Auto operating modes
- Snapshot capture to save signal events
- Side panel holds up to two snapshots
- One-tap playback to compare snapshots
- Optimized layout for the 3.5” Raspberry Pi touchscreen





System Prototyping and History

Establishing Communications

- Verified connectivity branch-by-branch (antenna → SDR → Raspberry Pi)
- Record .wav file from SDR stream → confirm audible signal playback
- Validate correct frequency tuning and gain settings

Radio Diversity Evaluation

- Extracted SNR (Signal-to-Noise Ratio) from each SDR channel
- Identified strongest signal paths and validated diversity gain

Automation and UI

- Automated SNR logging and visualization in GUI
- All branches established in wired setup

Audio Stream

- Added audio streaming from SDRs
- Audio comes from SDR focused by selection modes

Wireless Implementation

- Added wireless functionality to system
- Receiver Pis with SDR and antenna send audio and data to central Pi

Diversity Testing

- Test FM transmissions at varying ranges/angles
- Compared performance across diversity modes (selection, switching, etc.)

Testing: Spatial Diversity

Test Setup

- Three SDR units placed at varying distances from each other
- All tuned to the same target frequency
- Raspberry Pis synchronized to record SNR, RSSI, and timestamps
- Controlled environment with repeated signal transmissions

Measurements Collected

- SNR differences between receivers
- Packet/signal drop occurrences
- Variations in RSSI based on receiver position
- Time correlation of peaks and fades across SDRs

Expected Result

- One SDR will often have a stronger or more stable signal than the others
- Combining spatially separated receivers increases overall detection reliability

Results:

- All three SDR receivers successfully captured the transmitted signal
- Each receiver showed different SNR/RSSI levels based on its physical position
- At least one SDR consistently provided a strong, stable signal during every test
- No instances where all SDRs dropped or lost the signal at the same time
- Spatial separation improved overall detection reliability as expected
- Results confirm that multi-receiver spatial diversity works as designed

Testing: Frequency Diversity

Objective

- Test Frequency Diversity using two FM transmitters (88.1 MHz, 92.5 MHz).
- Verify automatic selection of strongest frequency.

Setup

- Two transmitters broadcasting identical audio.
- SDR collects SNR/RSSI for both frequencies.
- Diversity mode enabled on central unit.

Procedure

- Monitor 88.1 and 92.5 MHz simultaneously.
- Change transmitter positions and add obstacles.
- Observe automatic switching based on signal strength

Expected Behavior

- System picks the frequency with highest SNR/RSSI.
- Smooth audio transition when switching.
- Real-time updates shown on UI.

Results

- Automatic switching worked correctly.
- SNR/RSSI changes reflected instantly on UI.
- Audio remained stable after switching.

Testing: Time Diversity

Objective

- Test Time Diversity using one FM & one receiver
- Verify automatic selection of strongest frequency.

Setup

- One transmitter broadcasting messages of audio.
- SDR collects SNR/RSSI for both frequencies & captures snapshots of the audio messages

Procedure

- Take snapshots of the message at different time intervals over the span of a few minutes
- Change transmitter positions to affect signal quality
- Hit playback button and observe automatic playback based on signal strength at different times

Expected Behavior

- System picks the saved snapshot with highest SNR/RSSI.
- Mutes live audio while playing back snapshot

Results

- Automatic switching worked correctly.
- SNR/RSSI changes reflected instantly on UI.
- Audio was saved and played back at intervals

Performance Evaluation

Metric	How We Measured	Test Setup	Results
Communication Link	Record/playback audio stream	Transmitter close to SDR with no obstructions	Clear, audible signal with minimal distortion or cuts
Signal Strength (RSSI)	Read RSSI values from SDR drivers	Transmitter close to SDR with no obstructions. Streaming data to console.	Stable values above background noise floor
Signal-to-Noise Ratio (SNR)	Extract SNR per channel when antenna is near to transmitter	Transmitter close to SDR with no obstructions. Streaming data to console	Consistently > 35dB
Diversity Gain	Compare strongest SDR vs. combined diversity output	Spatial, Time, and Frequency Diversity tests detailed earlier.	Diversity consistently outperforms single SDR
Latency / Processing Time	Measure Pi's processing + GUI update delay	Pressing buttons on touchscreen during operation and filming.	Unfortunately not real-time (≤ 1 sec) responsiveness, but within 3 sec
Robustness Testing	Varied antenna positions & interference sources	Moving transmitter/antennas around lab (behind and under obstacles.	Reliable detection under different conditions

Challenges and Solutions

Challenges:

- Non-focused SDRs were not draining audio buffers
  - Audio latency grew extremely large (30–60+ seconds)
  - Large backlogs caused huge delays when switching SDRs
  - Latency compounded the longer the system ran
  - Wireless receivers accumulated even more delay due to jitter
  - Old “ghost audio” played even after SDRs disconnected
- Choppy Audio from wireless communications
  - Packets arriving out of order
  - Signal cutting every 500ms
- Touchscreen touch issues
  - Multiple touchscreens wouldn’t calibrate / register touch
  - One would even brick the system OS on bootup

Solutions:

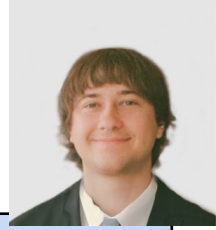
- Implemented bounded playback buffers for each SDR and trimmed buffers aggressively on SDR switching
  - Ensured all SDRs stay aligned to near real-time
  - Cleared buffers immediately on disconnect events
  - Reduced jitter warmup time for faster startup
  - Achieved consistent <10-second latency across all receivers
- Switched from UDP to TCP and added jitter buffer along with other measures to prevent audio loss
  - Absorbs startup and network jitter
  - Guarantees no lost or out of order packets
  - Queue overflow protection
- Finally, we found a touchscreen that was a plug-and-play option
  - No extra code or drivers
  - Uses USB to add touch functionality

Budget

Parts & Costs

- RTL-SDRs (x3) – ~$120
- Antennas (x3) – ~$45
- Coaxial Cables (x3, 10–20 ft) – ~$30
- Touchscreen Display – ~$50
- Walkie Talkie – ~$30
- Raspberry Pi 4 – Provided by UCF ($0)

Estimated Total: ~$275



Work Distribution

Member	Assembly	Administrative	Signal Processing	User Interface	Hardware System Design
Kim	Secondary • Secondary Team Lead • Assign Roles	Primary • Schedule Meetings • Correspond with Dr. Chan	Primary • Software Framework Of Diversity Alg.		
Jurgis				Primary • Design Main UI Layout Visuals	Secondary • Secure Screen With Touch Capabilities
Evan	Primary • Team Lead • Ensure Functional Device		Secondary • Wireless & Audio Implementation • Troubleshooting Issues	Secondary • .Wav File Saving And Playback for Snapshots • Frequency Switch for SDRs	Primary • Design & Assembly Of Hardware • Test All Components