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

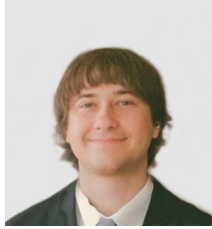
Objectives

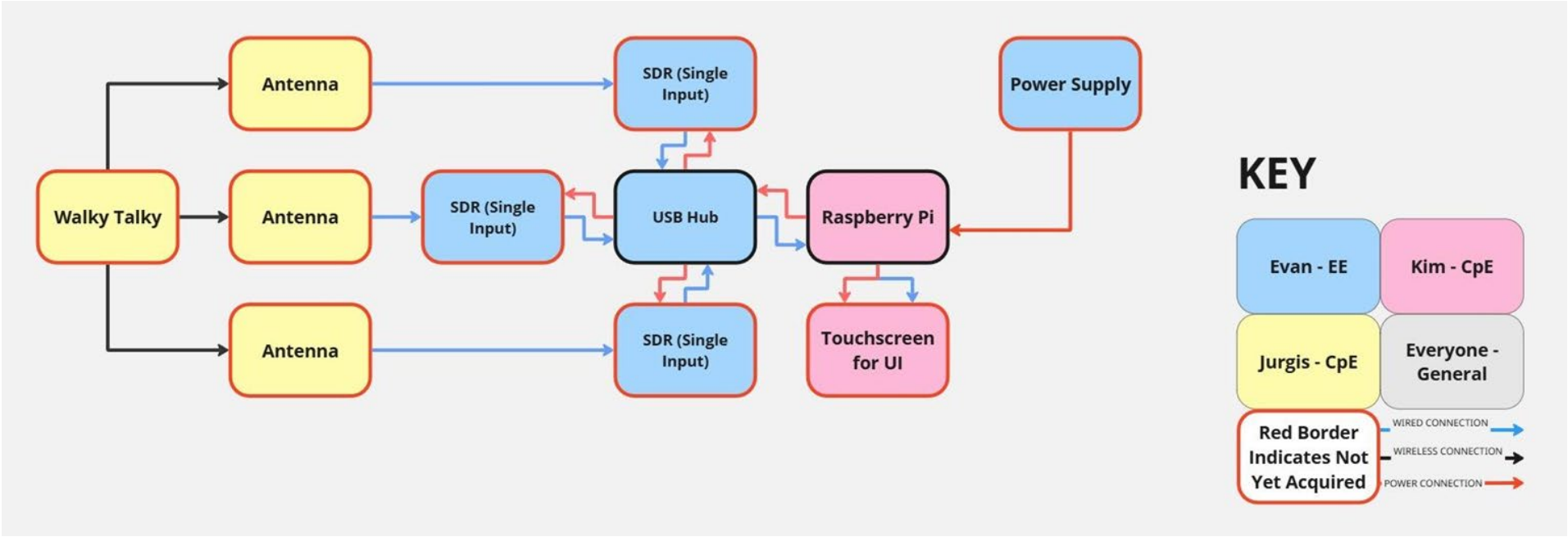
- Implement DSP algorithms to calculate SNR/RSSI per branch
- Apply hysteresis/smoothing to avoid unstable switching
- Achieve selection latency < 50 ms (parallel) / < 150 ms (multiplexed)
- Validate improvement through controlled lab/field tests

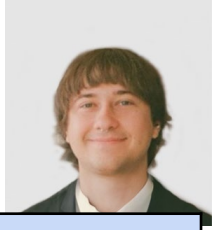
Engineering Specifications

PARAMETER	TARGET VALUE	NOTES
Total Cost	≤ $400	Amazon-sourced COTS components
Recieve Branches	3 antennas / receivers	Minimum required for diversity benefit
Operating Band	FM 88–108 MHz	Legal & accessible signals for proof-of-concept
Selection Latency	< 50 ms (parallel) / < 150 ms (multiplex)	Ensures smooth switching performance
Performance Gain	≥ 6 dB SNR improvement or 2× PER reduction	Demonstrates diversity benefit
Power Requirement	≤ 10 W	Can run on USB power bank

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

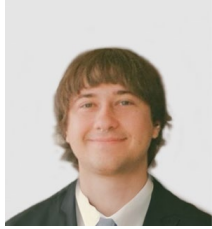
Connectivity & Expandability:

- 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

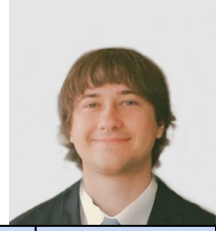


We selected 3 × RTL-SDR v3 units connected via a powered USB hub.

- 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 v3)	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)

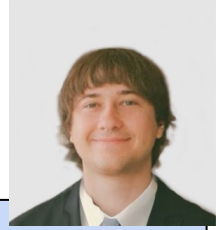
Hardware: Antenna Selection



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

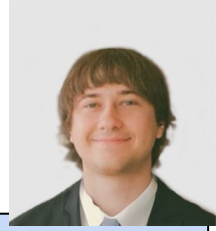
Key Selection Criteria

- 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

- **3.5″ Raspberry Pi-Compatible Touchscreen**
  - 480×320 resolution is sufficient for metrics/logs
  - Connects via GPIO/SPI, which frees HDMI & USB
  - Low power
  - Compact, portable, and panel-mount ready
- GUI Implementation
  - Python, optimized for small screens
  - Supports manual overrides & diagnostics
- Raspberry Pi 3 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



Software/Hardware: Broadcast Method

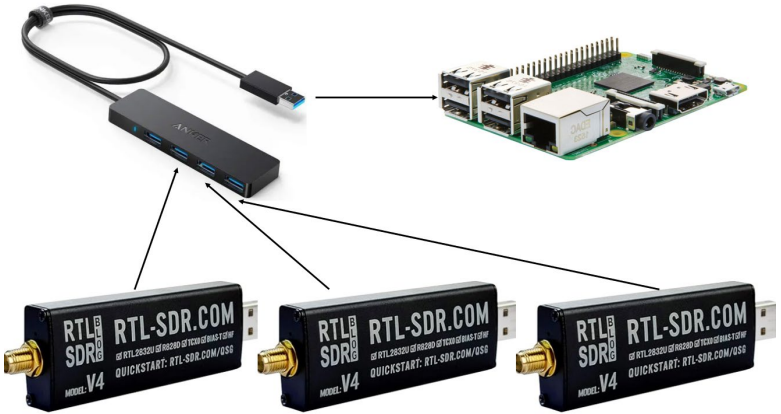
Off-the-shelf handheld walkie-talkies

- 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
Walkie-Talkie (Analog FM)	Off-the-shelf handheld radio	Readily available, long range voice-band signal	No digital data, limited control

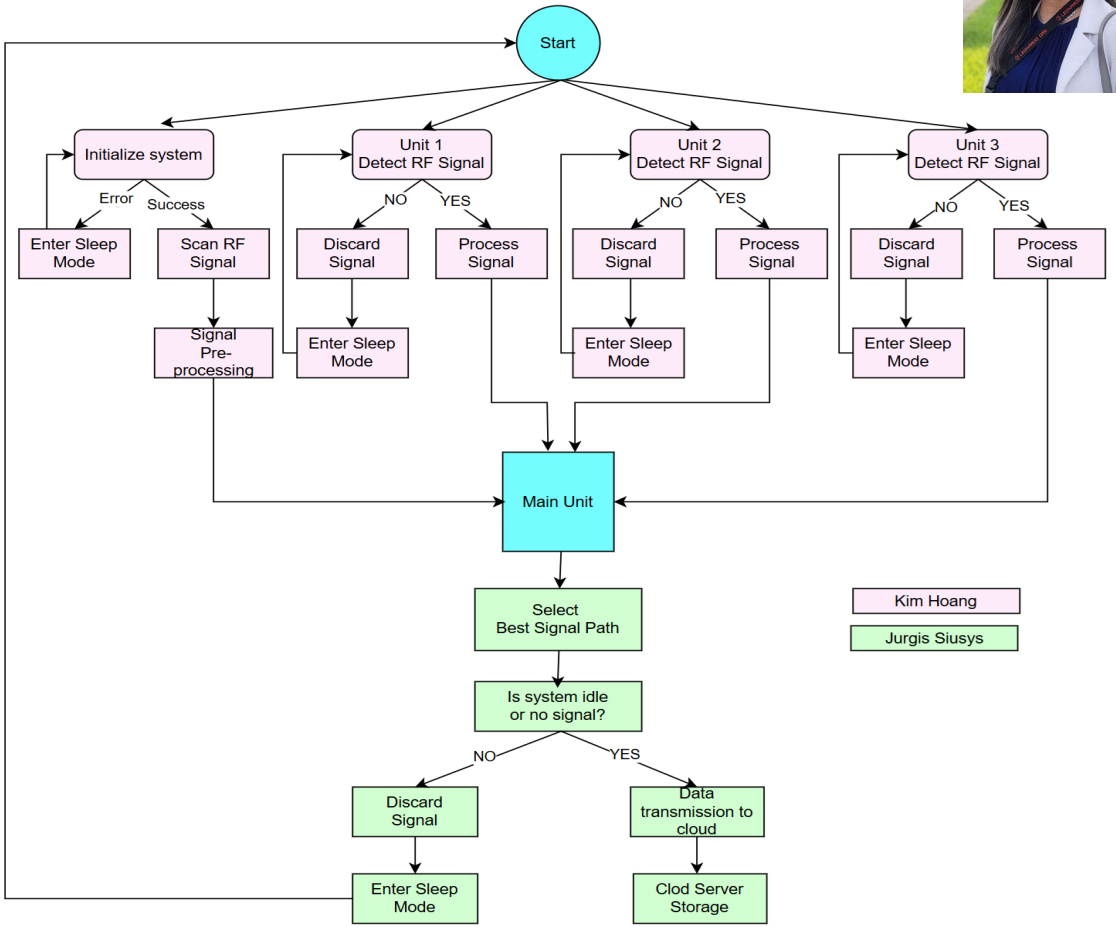
Hardware: Schematics

As one of the main foundations of our project is readily available, off the shelf products, there is no significant PCB requirement. Shown below is the block schematic for the whole system



Software Block Diagram

- System Initialization
- RF Signal Detection Units (Unit 1, 2, 3)
- Main Unit Integration
- Signal Path Selection
- Cloud Interaction



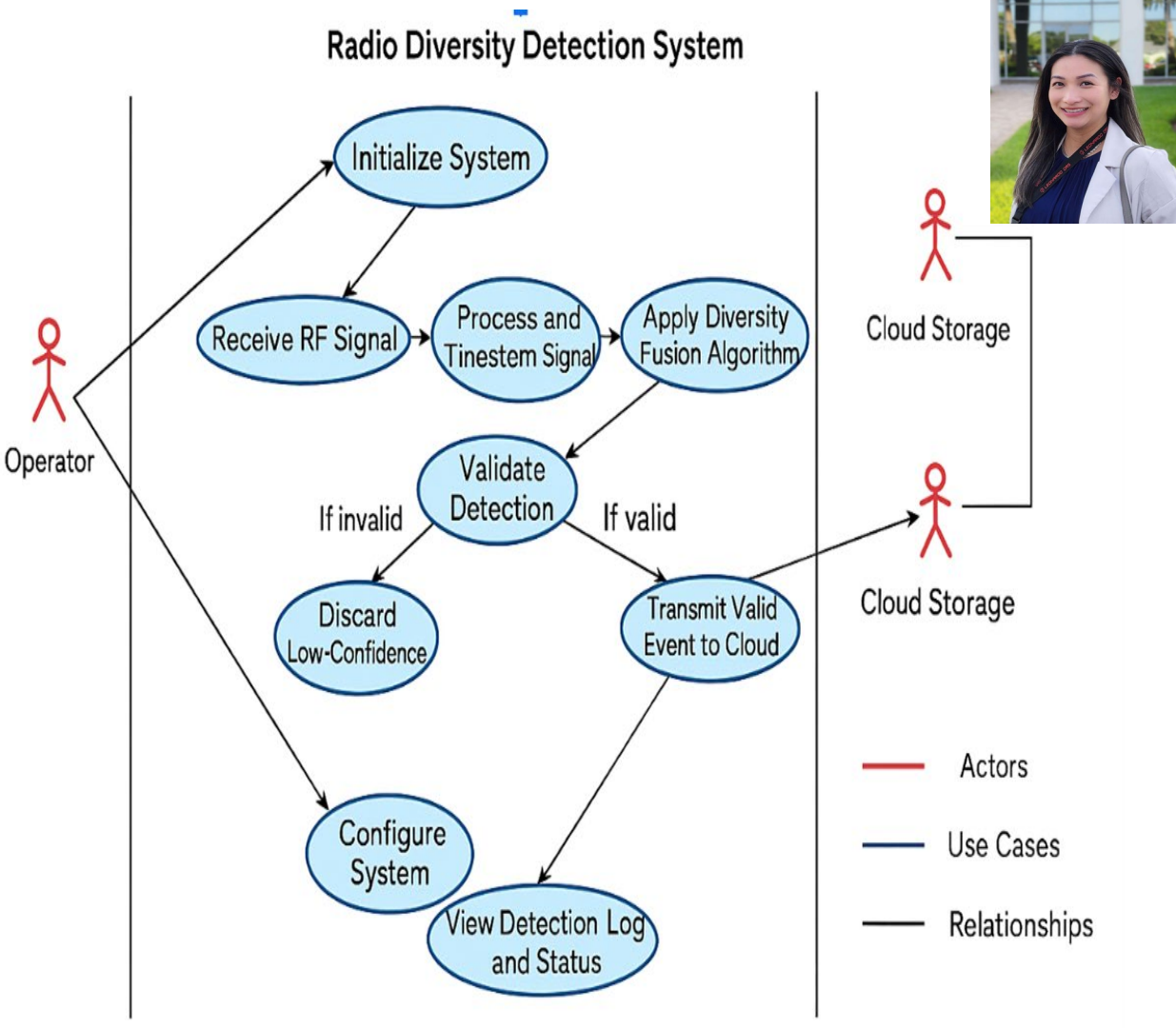
Software Design

Use Case Diagram

Operators trigger most use cases, including system initialization and configuration

Validation result branches determine whether the detection is stored or discarded

Transmitted events are pushed to Cloud Storage, where the cloud becomes an observer or receiver



Software Design

Startup & Signal Acquisition
System initializes and begins scanning for incoming RF signals.

Signal Threshold Check
Filters out low-confidence signals before further analysis.

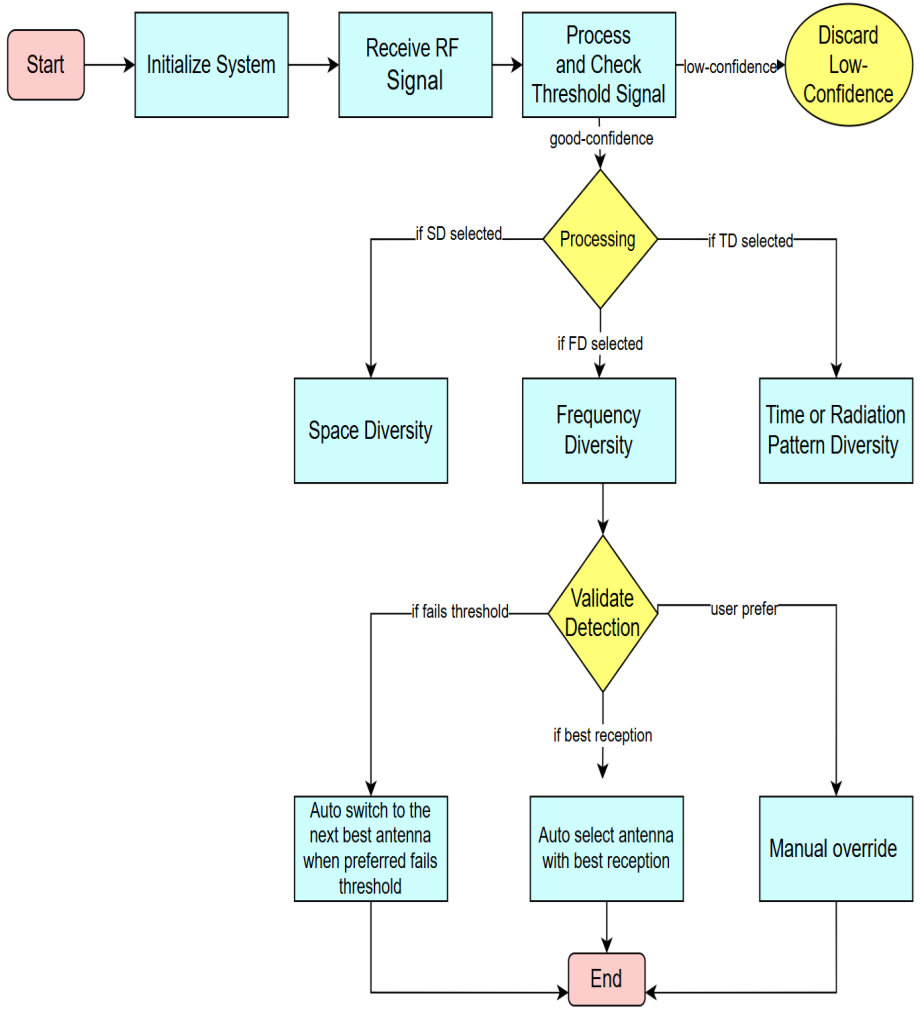
Apply Diversity Fusion
Good signals are processed using:

- Space Diversity
- Frequency Diversity
- Time/Radiation Pattern Diversity

Detection Validation & Decision Logic
Validated detections trigger:

- Automatic antenna switching or selection
- Manual override if needed

End of Processing Cycle
The system concludes the detection process and returns to idle or scanning mode.



Software Design

Start State: System begins in Idle Mode, awaiting signal activity.

Scanning Mode: System transitions to scan for RF signals in the environment.

Signal Detected?

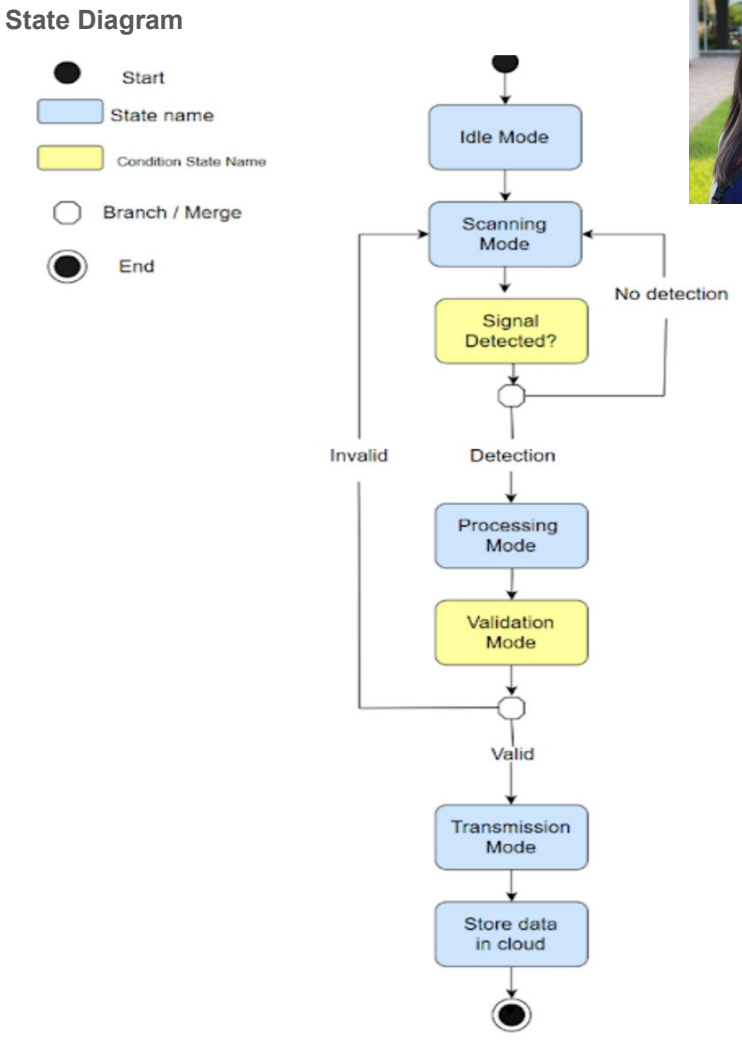
- If No Detection, system loops back to Scanning Mode.
- If Detection, proceed to Processing Mode.

Processing Mode: Performs signal analysis and preprocessing.
Validation Mode:

- If Invalid, system loops back to Scanning Mode.
- If Valid, transition to Transmission Mode.

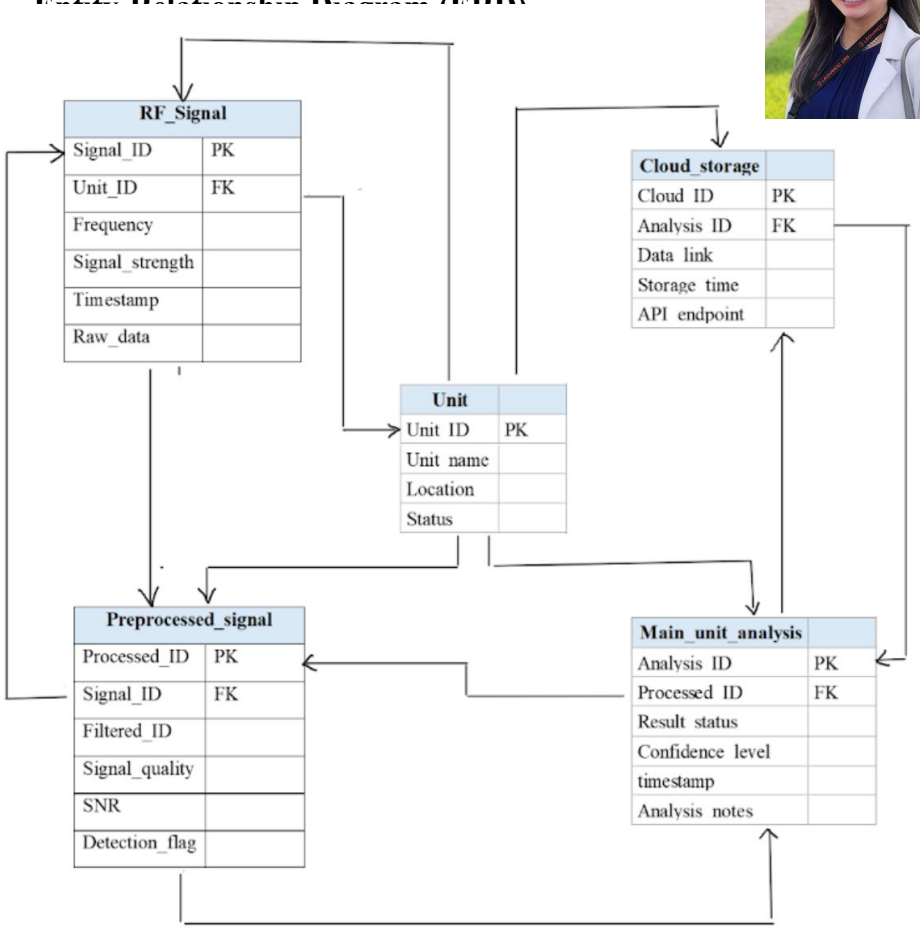
Transmission Mode: Valid signal data is prepared for cloud uplo
Store Data in Cloud: Final state where data is logged to cloud storage.

End State: Represents completion of one signal detection and transmission cycle.



Software Design

- **RF_Signal**: Stores raw RF data tied to a sensing unit.
- **Preprocessed_Signal**: Logs filtered signal quality and detection flags.
- **Main_Unit_Analysis**: Validates processed signals and stores results.
- **Cloud_Storage**: Holds final analysis for remote access.
- **Unit**: Tracks each node's ID, location, and status.



Software Design

Initialization: System begins when radio diversity is off.

Signal Acquisition & Preprocessing: Collects and preprocesses RF signals.

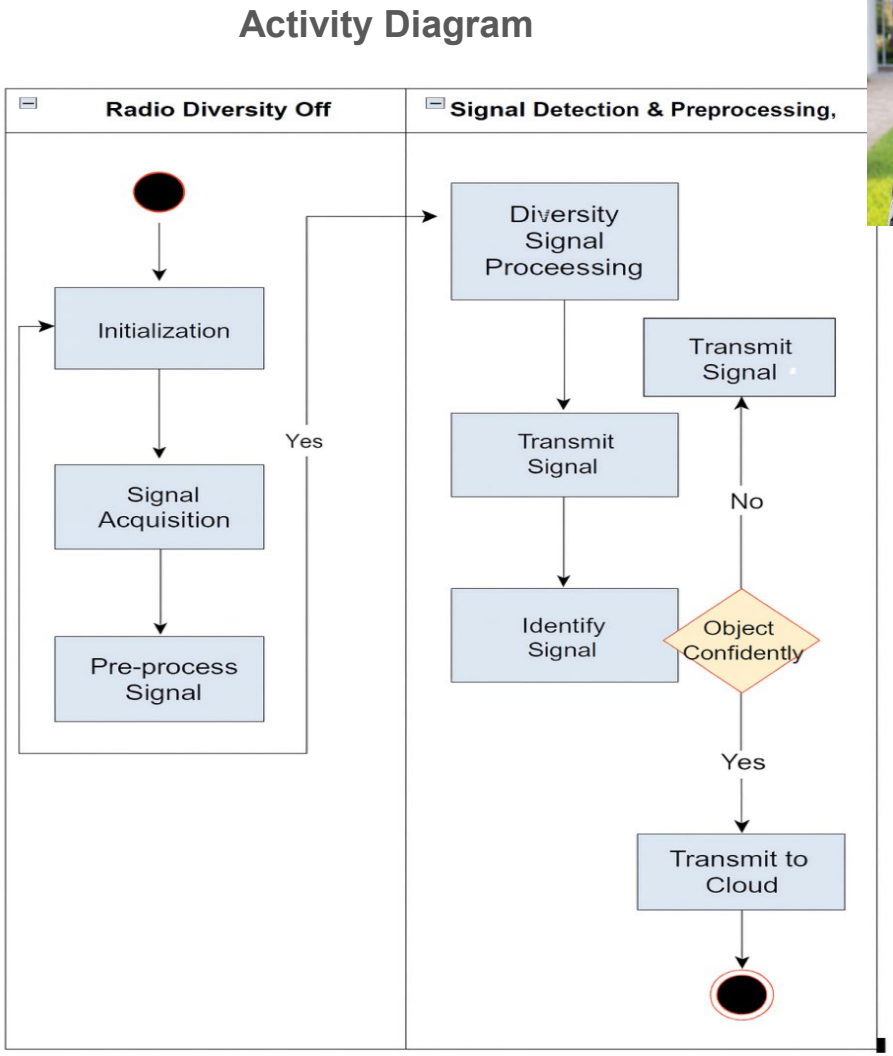
Diversity Signal Processing: Activated when signal confidence is sufficient.

Signal Identification: Determines if the signal can be confidently classified.

Confidence Evaluation:

- If **yes** → Transmit to cloud.
- If **no** → Return to transmit and reprocess.

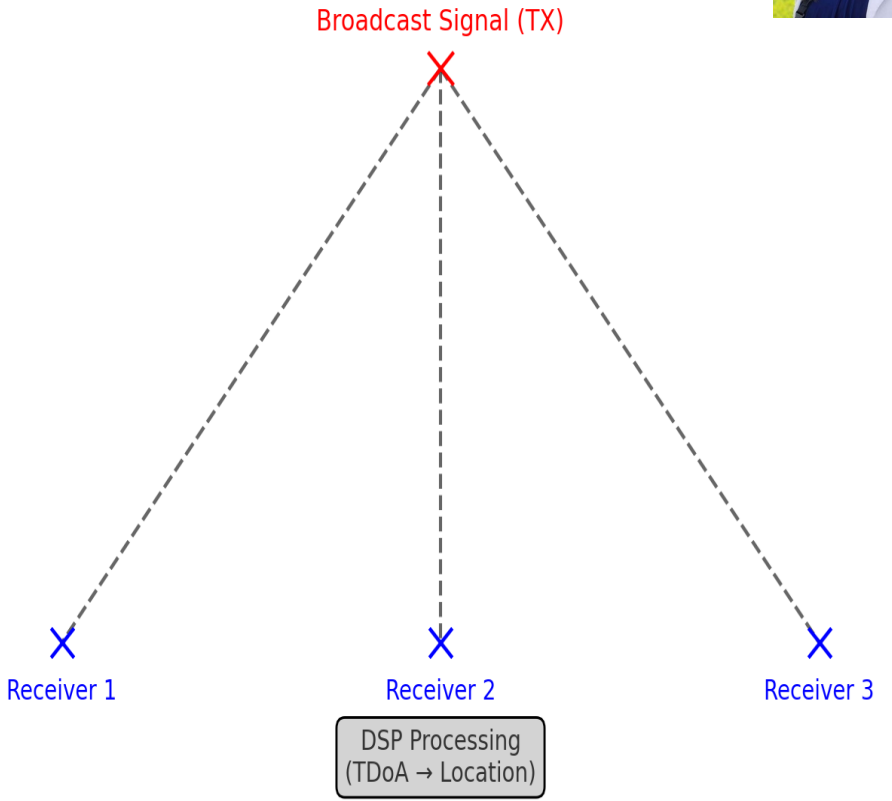
Cloud Transmission: Final step for storing validated signals.



Software Design

- Broadcast signal is received by multiple antennas
- Each receiver records the signal's arrival time
- DSP compares arrival times (TDoA)
- Time differences define hyperbolic curves
- Intersection of curves = estimated source location

Localization Diagram



Software Design

User Interface (UI) Design

UI Component	Description	Implementation
Dashboard	Displays the overall status of the radio diversity system, including node connectivity, signal integrity, and alert flags.	Web-based (React.js, HTML, JavaScript)
Live Detection Panel	Shows real-time RF signal strength, timestamps, and triangulated object location using visual maps or graphs.	WebSocket integration for live updates
Data Log Section	Provides access to historical logs of detected signal events, node activity, and processing metrics.	SQL Database + API Retrieval
Settings Panel	Enables users to configure detection thresholds, signal bands, communication intervals, and sync policies.	UI Forms (HTML, Python Tkinter)

Software: Language Selection

Python vs Other High-Level Languages

Feature / Capability	Python	MATLAB	JavaScript / Node.js	Java
Signal Processing Libraries	NumPy, SciPy, PyDSP	Native DSP functions (but commercial)	Limited	Available but less commonly used
Ease of Use	Very intuitive and readable syntax	High-level scripting	Easy for web use, not DSP-heavy	Verbose for rapid prototyping
Hardware Interface Support	GPIO, SPI, I2C via libraries (e.g., RPi.GPIO)	Mostly simulation	Weak hardware integration	JavaFX/Serial API possible, less direct
Cloud Communication	MQTT, HTTP/REST, WebSockets supported	Possible with toolboxes	Strong web API integration	Good but heavier
GUI Development	Tkinter, PyQt, web-based UIs	App Designer	Browser-based	JavaFX, Swing
Best Use Case	Signal fusion, cloud logic, UI	Academic DSP modeling	Lightweight dashboards	Enterprise-level apps

Software: Language Selection

C vs Other Low-Level Languages

Feature / Capability	Embedded C	C++	Rust	Assembly
Real-Time Performance	Very fast and deterministic	Also fast, but object overhead possible	Fast and safe, more complex	Fastest, but hard to maintain
Microcontroller Support	Native and widely supported	Supported (but may require subset)	Limited adoption in embedded	Very low-level
Interrupt Handling	Precise, reliable	Possible, slightly more complex	Yes, with safety guards	Yes, but manual
Memory Usage	Very low, highly optimized	Slightly more than C	Managed but larger footprint	Minimal
Development Difficulty	Moderate	Slightly more complex	Steep learning curve	Very hard
Best Use Case	RF signal detection, SPI/ADC interaction	Performance with some abstraction	Safety-critical systems (future scope)	Bitwise control at hardware level

Software: Dev Environment

C Development Environment Comparison

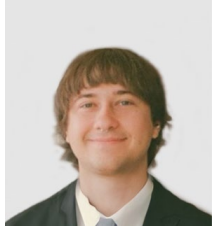
Environment / Tool	Strengths	Limitations
VS Code + GCC/ARM Toolchain	Cross-platform, integrates C/C++, good debugging, Git support	Requires manual toolchain configuration
PlatformIO (VS Code Extension)	Built-in support for embedded boards (ESP32, STM32, AVR, etc.)	Steeper learning curve, may be overkill for simple C projects
Eclipse + CDT	Mature IDE, integrated debugger, CMake support	Heavy and clunky UI, slower startup
Keil µVision	Great for ARM microcontrollers (e.g., STM32), excellent debugging tools	Windows-only, expensive for full version
Code::Blocks	Simple C IDE, low resource, fast startup	Lacks modern features and extensions
Geany / CLI (gcc/make)	Lightweight, fast on Raspberry Pi, full control over build	Minimal debugging, steeper learning for beginners

Software: Dev Environment

Python Development Environment Comparison

Environment / Tool	Strengths	Limitations
VS Code	Lightweight, great extensions for Python (Linting, Debugging, Jupyter)	Needs some setup for virtual environments and Git
PyCharm	Full-featured IDE, great for large projects, Django support	Heavier on resources, slow for simple scripts
Jupyter Notebook	Excellent for rapid prototyping, data visualization, interactive coding	Not ideal for structured, multi-file projects
Thonny	Beginner-friendly, very simple interface	Lacks advanced debugging and large project support
Raspberry Pi OS (with IDLE)	Built-in on Pi, easy to start, low resource	Basic editor, not ideal for debugging or Git
Anaconda (Spyder)	Good for scientific computing, prepackaged libraries	Heavy install, not tailored to embedded systems

Hardware/Software Testing



Establishing Communications

- Verify 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

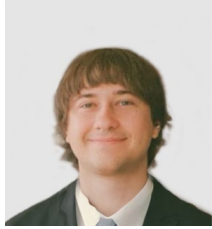
Radio Diversity Evaluation

- Extract SNR (Signal-to-Noise Ratio) from each SDR channel
- Compare performance across diversity modes (selection, switching, etc.)
- Identify strongest signal paths and validate diversity gain

Next Steps

- Automate SNR logging and visualization in GUI
- Test walkie-talkie transmissions at varying ranges/angles
- Validate system robustness under interference and multipath conditions

Performance Evaluation



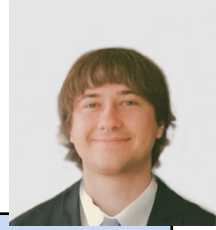
Metric	How We Measure	Success Indicator
Communication Link	Record/playback .wav file from SDR	Clear, audible signal with minimal distortion
Signal Strength (RSSI)	Read RSSI values from SDR drivers	Stable values above background noise floor
Signal-to-Noise Ratio (SNR)	Extract SNR per channel	≥ 10 dB improvement when diversity enabled
Diversity Gain	Compare strongest SDR vs. combined diversity output	Diversity consistently outperforms single SDR
Latency / Processing Time	Measure Pi's processing + GUI update delay	Real-time (≤ 1 sec) responsiveness
Robustness Testing	Vary antenna positions & interference sources	Reliable detection under different conditions

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 3 – Provided by UCF ($0)

Estimated Total: ~$275



Work Distribution

Member	Assembly	Signal Processing	User Interface	Hardware System Design
Kim	Secondary • Secondary Team Lead • Assign Roles	Secondary • Verify Detection Algorithm	Primary • Design Localization Algorithm	
Jurgis		Primary • Coding & Framework of Detection Alg	Secondary • Verify and Test Localization Algorithm	
Evan	Primary • Team Lead • Ensure Functional Device			Primary • Design & Assembly of PSU • Connect all Component to Needed Power

Plan for Completion

- Order and set up hardware (SDRs, antennas, cables, touchscreen)
- Configure Raspberry Pi and software environment
- Develop DSP algorithm for link selection
- Test system with controlled signals (walkie-talkie / FM)
- Analyze results and finalize report + presentation