KrakenSDR Passive Radar System

Technical Reference

Simon Knight

Independent Researcher, Adelaide, Australia

March 2026 • Version 2.0

Abstract. This document describes the design, implementation, and operation of a multi-beam passive radar system built on the KrakenSDR five-channel coherent receiver. The system exploits existing broadcast transmitters (FM, DAB, DVB-T) as illumination sources and processes the reflected signals to detect, track, and geolocate airborne targets. A distributed architecture streams IQ samples from a Raspberry Pi acquisition node to a workstation, where four parallel beams are processed through a pipeline of Wiener clutter cancellation, FFT-based cross-ambiguity function computation, CA-CFAR detection, Kalman filtering, and geographic projection. Detections are correlated with ADS-B aircraft positions for validation and recorded to HDF5 for offline analysis. Advanced signal intelligence algorithms—CLEAN sidelobe suppression, MVDR spatial filtering, and micro-Doppler extraction—extend the basic pipeline. A Dash web dashboard and an emerging Rust terminal UI provide real-time visualisation of Range-Doppler maps, target tracks, and geographic positions.

Contents

1 Introduction

1.1 What This System Is

This system implements *bistatic passive radar*—a surveillance technique that exploits existing broadcast transmitters as illumination sources rather than emitting its own signal. A five-channel KrakenSDR receiver captures both a reference signal (direct path from the transmitter) and four surveillance signals (reflections from targets). The system processes these signals in real time to produce Range-Doppler detections, maintain persistent target tracks via Kalman filtering, and project those tracks onto a geographic map.

Unlike active radar, passive radar does not reveal its presence to radar warning receivers and incurs no transmitter cost. The tradeoff is more complex signal processing: the reference signal must be separated from the surveillance signal, and the bistatic geometry introduces range ambiguity that active radar avoids.

1.2 Design Philosophy

Two principles govern the system's architecture:

1. **Distributed acquisition, centralised processing.** The Raspberry Pi handles only data acquisition and streaming; all compute-intensive DSP runs on a workstation with sufficient CPU headroom for four parallel beams at 65–80 frames per second.
2. **Per-beam independence.** Each surveillance channel is processed independently through an identical pipeline. This enables straightforward parallelism via `ProcessPoolExecutor` and avoids the complexity of cross-beam fusion until the system matures.

1.3 Who Should Read This Document

- **Radar engineers** evaluating the system for passive radar research or deployment, needing to understand the signal processing pipeline and its limitations.
- **Software engineers** extending the system with new algorithms, hardware backends, or visualisation modes.
- **Operators** deploying and tuning the system in the field, needing to understand configuration, calibration, and diagnostic tools.

1.4 How to Read This Document

Section 2 provides the minimum radar theory needed to follow the design. Sections 3–4 cover the data model and system architecture. Sections 5–6 cover the core signal processing and tracking pipeline in depth. Section 8 describes the advanced DSP algorithms (CLEAN, MVDR, micro-Doppler). Sections 7–11 cover integration, visualisation, and deployment. Readers wanting to get started quickly should read §4 and §11, then refer to individual algorithm sections as needed.

2 Background: Passive Radar Fundamentals

2.1 Bistatic Geometry

In bistatic radar, the transmitter and receiver are at different locations. A target at position $\mathbf{p}$ produces a reflected signal that travels a total *bistatic range*:

$$R_b = |\mathbf{p} - \mathbf{t}| + |\mathbf{p} - \mathbf{r}| - |\mathbf{t} - \mathbf{r}| \tag{1}$$

where $\mathbf{t}$ is the transmitter position, $\mathbf{r}$ is the receiver position, and the baseline $|\mathbf{t} - \mathbf{r}|$ is subtracted to give the *excess* path length. The locus of constant bistatic range is an ellipsoid with the transmitter and receiver as foci.

> **Bistatic range ambiguity**
>
> Unlike monostatic radar where range maps to a unique sphere, a bistatic range value maps to an ellipsoid. A single receiver beam constrains the target to the intersection of the ellipsoid and the beam cone, but the resulting position estimate has significant uncertainty without additional information (a second receiver, altitude assumptions, or ADS-B correlation).

2.2 Cross-Ambiguity Function

The cross-ambiguity function (CAF) measures the similarity between the reference signal $r(t)$ and a time-delayed, frequency-shifted copy of the surveillance signal $s(t)$:

$$\mathrm{CAF}(\tau, f_d) = \int_0^T s(t)\, r^*(t - \tau)\, e^{-j2\pi f_d t}\, dt \tag{2}$$

where τ is the time delay (proportional to bistatic range) and f_d is the Doppler shift (proportional to target radial velocity). Computing the CAF for all (τ, f_d) pairs yields a two-dimensional *Range-Doppler map* in which targets appear as peaks.

2.3 Clutter and Direct-Path Interference

The strongest signal component in the surveillance channel is the direct-path signal from the transmitter—the same signal as the reference channel, but received via the surveillance antenna's sidelobe. This "zero-Doppler clutter" must be cancelled before weak target echoes become visible. The system uses adaptive Wiener filtering (Section 5.1) to suppress this interference.

3 Data Model

3.1 IQ Header (1024 bytes)

Every frame from the Heimdall DAQ firmware begins with a 1024-byte binary header (`_receiver/iq_header.py`). The header uses little-endian byte order to match Python's `struct.pack` native format. Key fields:

Field	Type	Description
`sync_word`	`uint32`	Magic value `0x2bf7b95a` for frame synchronisation
`frame_type`	`uint32`	0=DATA, 1=DUMMY, 2=RAMP, 3=CAL, 4=TRIGW
`active_ant_chs`	`uint32`	Number of active channels (typically 5)
`rf_center_freq`	`uint64`	Centre frequency in Hz
`adc_sampling_freq`	`uint64`	ADC sampling rate in Hz
`cpi_length`	`uint32`	Samples per Coherent Processing Interval
`time_stamp`	`uint64`	Unix epoch (auto-detected units)
`adc_overdrive_flags`	`uint32`	Per-channel clipping indicators
`if_gains[32]`	`uint32[32]`	Per-channel gain values
`noise_source_state`	`uint32`	Calibration noise source on/off

Table 1. Key IQ header fields. The full header occupies 1024 bytes; unused fields are reserved for firmware extensions.

3.2 Detection

A detection is a discrete peak extracted from the Range-Doppler map by the CA-CFAR algorithm (Section 5.3):

Listing 1. Detection data class

```
@dataclass
class Detection:
    range_bin: int           # Integer bin index
    doppler_bin: int         # Integer bin index
    range_bin_f: float       # Fractional (sub-bin interpolation)
    doppler_bin_f: float     # Fractional (sub-bin interpolation)
    range_m: float           # Bistatic range in metres
    doppler_hz: float        # Doppler shift in Hz
    snr_db: float            # Signal-to-noise ratio in dB
    noise_estimate: float    # Estimated noise floor at this cell
    timestamp: float         # Unix epoch (seconds)
    beam_id: int             # Beam index (0--3)
```

3.3 Track

A track represents a persistent target identity maintained across frames by the Kalman filter and data association pipeline (Section 6):

Listing 2. Track data class (key fields)

```
@dataclass
class Track:
    track_id: str            # "B0-001" (beam, sequence)
    beam_id: int
    status: TrackStatus      # TENTATIVE / CONFIRMED / COASTING
    kf: KalmanFilter         # 4D state: [range, range_rate,
                             #            doppler, doppler_rate]
    age: int                 # Frames since creation
    hits: int                # Total detections associated
    consecutive_misses: int  # Misses before deletion
    lat: float               # Geographic latitude
    lon: float               # Geographic longitude
```

```
    adsb_correlated: bool      # Matched to ADS-B aircraft
    flight_callsign: str       # ADS-B callsign (if correlated)
```

3.4 Range-Doppler Matrix

The primary data product is a two-dimensional matrix of shape (n_doppler, n_range), stored as `float32` values in dB:

- **Doppler axis:** bin 0 corresponds to $-f_s/2$ (maximum negative velocity); the centre bin corresponds to zero Doppler (stationary clutter); the last bin corresponds to $+f_s/2$.
- **Range axis:** bin 0 is zero bistatic range (direct path); the last bin corresponds to the maximum configured bistatic range.
- **Storage:** one `.npy` file per beam per frame in `output/beam_N/`.

4 Architecture

4.1 System Topology

The system comprises two physical nodes connected over a local network:

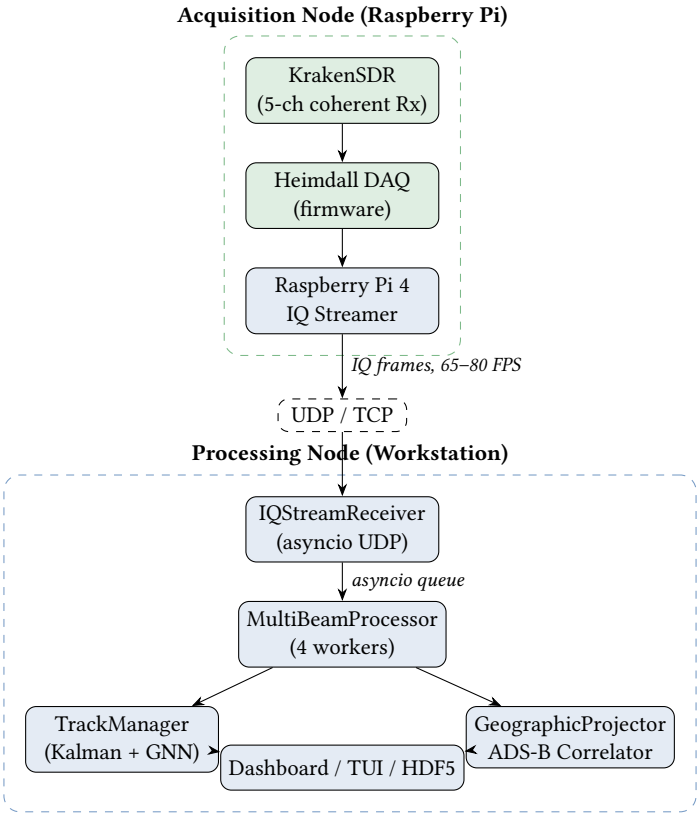

Figure 1. System topology. The acquisition node handles only IQ capture and streaming; all signal processing runs on the workstation.

4.2 Per-Frame Processing Pipeline

Each IQ frame traverses the following stages:

1. **Receive:** `IQStreamReceiver` accepts a UDP datagram, parses the 1024-byte IQ header, and places the frame on an asyncio queue.

2. **Dispatch:** `MultiBeamProcessor` distributes four beam jobs to a `ProcessPoolExecutor` (one worker per beam).
3. **Per-beam DSP** (in worker process):
   (a) Wiener SMI-MRE clutter cancellation (§5.1)
   (b) Hann windowing
   (c) FFT-based cross-ambiguity function (§5.2)
   (d) (Optional) CLEAN sidelobe suppression (§8.1)
   (e) CA-CFAR detection (§5.3)
4. **Track update** (main process): Kalman prediction, GNN association, M-of-N confirmation (§6).
5. **Projection:** bistatic range and beam azimuth to geographic coordinates (§7).
6. **Correlation:** match tracks to ADS-B aircraft (§7.2).
7. **Output:** update dashboard, record to HDF5, stream to TUI.

4.3 Concurrency Model

Thread environment variables

The system sets `OMP_NUM_THREADS=2`, `OPENBLAS_NUM_THREADS=2`, and `MKL_NUM_THREADS=2` *before* importing NumPy. With four workers × two threads each, this yields eight total threads—manageable on modern hardware. Without this control, NumPy and FFTW can spawn dozens of threads per worker, causing severe contention.

The main process is single-threaded (asyncio event loop) and handles:

- Frame reception (UDP)
- Track management (Kalman updates are fast: <1 ms per beam)
- Geographic projection and ADS-B correlation
- Dashboard serving (Dash callbacks)

CPU-intensive DSP (Wiener filtering, FFT, CFAR) runs in worker processes to bypass the GIL.

4.4 Project Layout

Directory	Contents
`_receiver/`	Data acquisition: IQ header parsing, UDP receiver, service discovery, connection monitoring
`_signal_processing/`	DSP pipeline: multi-beam processor, Wiener filter, CFAR, tracking, CLEAN, MVDR, micro-Doppler
`_integration/`	External systems: ADS-B client, HDF5 recording/playback, TUI WebSocket server
`_UI/_web_interface/`	Dash web dashboard: Range-Doppler grid, map, metrics, controls
`config/`	Pydantic configuration models
`tui/`	Rust terminal UI (ratatui + Kitty graphics)
`dsp-core/`	Rust DSP foundation (rustfft, nalgebra, ndarray)
`run_*.py`	Entry points (server, headless, dashboard, Pi streamer)

Table 2. Top-level project layout.

5 Signal Processing Pipeline

5.1 Wiener Clutter Cancellation

5.1.1 Problem

The surveillance channel receives the target echo $s_{\text{target}}(t)$ plus a much stronger direct-path copy of the transmitted signal $\alpha \cdot r(t)$, where $r(t)$ is the reference signal. The direct-path component typically exceeds the target echo by 40–60 dB. Without cancellation, targets are invisible.

5.1.2 Design Options

Three clutter cancellation approaches were considered:

1. **Batch Wiener filter (LMS):** slow convergence, sensitive to non-stationarity.
2. **Extensive Cancellation Algorithm (ECA):** block-based, requires computing large covariance matrices.
3. **Wiener SMI-MRE:** single-pass sample matrix inversion with minimum redundancy estimation. Fast, effective for the single-reference case.

5.1.3 Decision

Wiener SMI-MRE was chosen (via the pyapril library) because it provides adequate cancellation (30–50 dB suppression) in a single matrix operation per CPI, with latency under 10 ms per beam.

5.1.4 Algorithm

Given reference signal $\mathbf{r} \in \mathbb{C}^N$ and surveillance signal $\mathbf{x} \in \mathbb{C}^N$, the filter coefficient is:

$$\hat{\alpha} = \frac{\mathbf{r}^H \mathbf{x}}{\mathbf{r}^H \mathbf{r} + \epsilon} \tag{3}$$

The clutter-cancelled signal is:

$$\mathbf{x}_{\text{clean}} = \mathbf{x} - \hat{\alpha}\, \mathbf{r} \tag{4}$$

> **pyapril implementation**
>
> The actual implementation in pyapril uses a multi-tap filter with K taps to model the multipath structure of the direct-path interference. The single-tap description above captures the essential idea; the multi-tap version replaces $\hat{\alpha}$ with a vector $\hat{\boldsymbol{\alpha}} \in \mathbb{C}^K$ estimated via sample matrix inversion.

5.1.5 Consequences

- Effective for stable transmitter signals (FM, DAB).
- Performance degrades if the reference signal itself is corrupted by multipath.
- No real-time adaptation: the filter is recomputed per CPI, not per sample.

5.2 Cross-Ambiguity Function Computation

5.2.1 Problem

After clutter cancellation, the system must compute the CAF (Equation 2) across all range-Doppler bins to produce the two-dimensional detection surface.

5.2.2 Implementation

The system uses `pyapril`'s `cc_detector_ons` function, which implements an FFT-based batched cross-correlation:

Algorithm 1 FFT-based Range-Doppler map generation

Require: Reference $\mathbf{r}$, surveillance $\mathbf{x}$, N_d Doppler bins, N_r range bins
Ensure: Range-Doppler matrix $\mathbf{M} \in \mathbb{R}^{N_d \times N_r}$

$\mathbf{R} \leftarrow \text{FFT}(\mathbf{r},\ n = N_d)$
for $\ell = 0$ **to** $N_r - 1$ **do**
  $\mathbf{x}_\ell \leftarrow \mathbf{x}[0 : N - \ell]$ ▷ delay by ℓ samples
  $\mathbf{S}_\ell \leftarrow \text{FFT}(\mathbf{x}_\ell,\ n = N_d)$
  $\mathbf{C}_\ell \leftarrow \mathbf{R}^* \odot \mathbf{S}_\ell$ ▷ element-wise cross-correlation
  $\mathbf{M}[:, \ell] \leftarrow |\text{IFFT}(\mathbf{C}_\ell)|$
end for
$\mathbf{M} \leftarrow 10 \log_{10}(\mathbf{M} + \epsilon)$ ▷ convert to dB

The Doppler resolution is $\Delta f_d = f_s / N_d$ and the range resolution is $\Delta R = c/(2 f_s)$, where f_s is the sampling frequency and c is the speed of light.

5.2.3 Performance

Typical latency: 30–60 ms per beam for $N_d = 2048$, $N_r = 128$. This is the computational bottleneck of the pipeline. NumPy's FFT is JIT-compiled via Numba in pyapril for additional speed.

5.3 CA-CFAR Detection

5.3.1 Problem

The Range-Doppler map contains target peaks embedded in a non-uniform noise floor. A fixed threshold would either miss weak targets (threshold too high) or produce excessive false alarms (threshold too low). CFAR adapts the threshold to the local noise environment.

5.3.2 Algorithm

The system uses two-dimensional Cell-Averaging CFAR. For each cell under test (CUT) at position (d, r):

1. A rectangular *guard region* of $G_r \times G_d$ cells around the CUT is excluded (to avoid target energy biasing the noise estimate).
2. A surrounding *training region* of $T_r \times T_d$ cells provides the noise floor estimate $\hat{P}_n$ = mean(training cells).
3. The detection threshold is $\eta = \hat{P}_n \cdot \alpha_{\text{CFAR}}$, where α_{CFAR} is computed from the desired probability of false alarm P_{fa}:
$$\alpha_{\text{CFAR}} = N_{\text{train}} \left(P_{\text{fa}}^{-1/N_{\text{train}}} - 1 \right) \tag{5}$$
4. If CUT $> \eta$, a detection is declared.

5.3.3 Sub-Bin Interpolation

After peak detection, three-point parabolic interpolation refines the peak location to sub-bin accuracy:

$$\delta = \frac{y_{-1} - y_{+1}}{2(y_{-1} - 2y_0 + y_{+1})} \tag{6}$$

where y_{-1}, y_0, y_{+1} are the log-magnitude values at the peak and its neighbours. The fractional offset δ is clipped to $[-0.5, +0.5]$.

5.3.4 Zero-Doppler Suppression

The five bins centred on zero Doppler are blanked before CFAR processing to suppress residual clutter that survives Wiener cancellation. This prevents stationary objects (buildings, terrain) from saturating the detection list.

Parameter	Default	Description
Guard cells (range, Doppler)	3, 3	Exclusion zone around CUT
Training cells (range, Doppler)	10, 10	Noise estimation annulus
P_{fa}	10^{-6}	Target false alarm rate
Zero-Doppler bins	5	Blanked stationary clutter bins
Max detections	50	Per beam, ranked by SNR

Table 3. CA-CFAR default parameters.

6 Target Tracking

6.1 Track Lifecycle

Target tracks follow a three-state lifecycle:

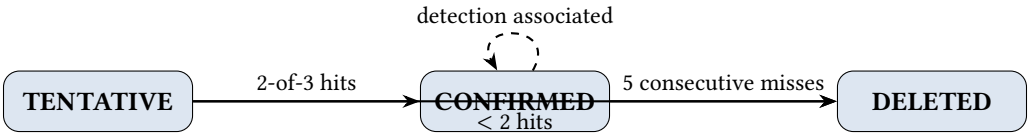

Figure 2. Track lifecycle state machine. Tracks begin as TENTATIVE and must accumulate sufficient hits (M-of-N confirmation) to become CONFIRMED.

6.2 Kalman Filter Model

The system uses a four-dimensional constant-velocity Kalman filter with state vector $\mathbf{x} = [\text{range}, \dot{r}, f_d, \dot{f}_d]^T$:

$$\mathbf{F}(\Delta t) = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{7}$$

The measurement vector $\mathbf{z} = [\text{range}_m, f_d]^T$ comes from the CFAR detector. The process noise matrix $\mathbf{Q}$ uses a continuous white noise acceleration model scaled by a tuneable parameter q (default 0.5).

6.3 Data Association: Global Nearest Neighbour

Detections are associated with existing tracks using the Global Nearest Neighbour (GNN) algorithm:

1. Compute a cost matrix $\mathbf{C}$ where C_{ij} is the Mahalanobis distance between track i's predicted measurement and detection j:

$$D_M = (\mathbf{z}_j - \hat{\mathbf{z}}_i)^T \mathbf{S}_i^{-1} (\mathbf{z}_j - \hat{\mathbf{z}}_i) \tag{8}$$

   where $\mathbf{S}_i = \mathbf{H}\mathbf{P}_i^- \mathbf{H}^T + \mathbf{R}$ is the innovation covariance.

2. Apply a gate: set $C_{ij} = \infty$ if $D_M > 9.21$ (99% χ^2 threshold for 2 degrees of freedom).
3. Solve the assignment problem via the Hungarian algorithm (`scipy.optimize.linear_sum_assignment`), which finds the optimal one-to-one matching in $O(n^3)$.
4. Unassociated detections initialise new TENTATIVE tracks.

7 Geographic Projection and ADS-B Integration

7.1 Bistatic Range to Geographic Coordinates

Given a confirmed track with bistatic range R_b (metres) and beam azimuth θ (degrees from true north), the system projects the target position onto a geographic coordinate:

1. The receiver position (ϕ_r, λ_r) and beam azimuth θ define a bearing from the receiver.
2. The range R_b defines the distance along that bearing.
3. The Vincenty direct formula computes the destination point: $(\phi_t, \lambda_t) = \text{destination}(\phi_r, \lambda_r, \theta, R_b)$.

Accuracy limitations

This projection assumes the target lies along the beam axis, which introduces angular error of $\pm 2{-}5^\circ$ due to antenna array mechanical tolerance and beam width. The bistatic range itself has $\pm 10{-}20\%$ error depending on clutter cancellation quality. ADS-B correlation (Section 7.2) provides ground truth for validating these estimates.

7.2 ADS-B Correlation

The system fetches aircraft positions from a local dump1090 instance via HTTP and correlates them with radar tracks:

1. **Spatial gating:** only consider ADS-B aircraft within 5 km of the projected track position.
2. **Velocity consistency:** check that the ADS-B ground speed and heading are consistent with the radar Doppler and beam geometry (within 10% tolerance).
3. **One-to-one assignment:** each radar track is matched to at most one aircraft.
4. **Enrichment:** matched tracks are annotated with the aircraft callsign, ICAO24 hex code, barometric altitude, and aircraft type.

This correlation serves two purposes: operational enrichment (operators see callsigns instead of anonymous track IDs) and system validation (comparing radar range/Doppler estimates against known ADS-B positions).

8 Advanced DSP Algorithms

8.1 CLEAN Sidelobe Suppression

8.1.1 Problem

Strong targets spread energy across the Range-Doppler map via the ambiguity function sidelobes of the transmitted waveform. Weak targets that fall within these sidelobes become invisible.

8.1.2 Algorithm

The CLEAN algorithm iteratively subtracts the strongest peak's sidelobe pattern:

Algorithm 2 CLEAN sidelobe suppression

Require: R-D map $\mathbf{M}$, ambiguity function $\mathbf{A}$, loop gain γ, max iterations K, threshold T_{dB}
Ensure: Cleaned R-D map $\mathbf{M}'$

$\mathbf{M}' \leftarrow \mathbf{M}$
for $k = 1$ **to** K **do**
  $(d^*, r^*) \leftarrow \arg\max_{d,r} |\mathbf{M}'(d, r)|^2$
  $\text{SNR} \leftarrow 10 \log_{10}\big(|\mathbf{M}'(d^*, r^*)|^2 / \text{median}(|\mathbf{M}'|^2)\big)$
  if $\text{SNR} < T_{\text{dB}}$ **then break**
  end if
  $\mathbf{M}' \leftarrow \mathbf{M}' - \gamma \cdot \text{shift}(\mathbf{A}, d^*, r^*) \cdot \mathbf{M}'(d^*, r^*)$
end for

Default parameters: $\gamma = 0.5$, $K = 5$, $T_{\text{dB}} = 15$. Typical latency: 20 ms per beam.

8.2 MVDR Spatial Filtering

8.2.1 Problem

Directional interference sources (cell towers, other transmitters) can produce persistent false detections. MVDR (Minimum Variance Distortionless Response) beamforming adaptively nulls these interferers while preserving the signal from the desired look direction.

8.2.2 Algorithm

Given multi-channel IQ data $\mathbf{X} \in \mathbb{C}^{N_c \times N_s}$ and a steering vector $\mathbf{a} \in \mathbb{C}^{N_c}$ toward the desired direction:

$$\mathbf{w}_{\text{MVDR}} = \frac{\mathbf{R}^{-1}\mathbf{a}}{\mathbf{a}^H \mathbf{R}^{-1}\mathbf{a}} \tag{9}$$

where $\mathbf{R} = \frac{1}{N_s}\mathbf{X}\mathbf{X}^H + \epsilon\mathbf{I}$ is the regularised spatial covariance matrix. The output signal is $y(t) = \mathbf{w}^H \mathbf{x}(t)$.

Diagonal loading $\epsilon = 10^{-3}$ provides numerical stability and robustness to steering vector errors. Typical latency: <5 ms per beam.

8.3 Micro-Doppler Extraction

Targets with rotating or oscillating components (propellers, helicopter rotors) exhibit modulation sidebands around the main Doppler peak. The `MicroDopplerExtractor` maintains a rolling spectrogram of the Doppler slice around each tracked target (±20 bins, 50 frame history). This signature can distinguish aircraft types—fixed-wing propeller aircraft produce a characteristic blade-flash pattern, while rotorcraft produce a wider, asymmetric modulation.

9 Recording and Playback

9.1 HDF5 Schema

Detections, track snapshots, and ADS-B data are recorded to HDF5 files with the following structure:

HDF5 Path	Contents
`/index/frames`	Frame index: timestamp, beam ID, detection/track start indices and counts
`/data/detections`	Flat table: timestamp, beam, range, Doppler, SNR
`/data/tracks`	Flat table: timestamp, beam, track ID, status, range, Doppler, lat, lon
`/adsb`	ADS-B snapshots: hex, callsign, lat, lon, altitude
`(attributes)`	Metadata: receiver/transmitter locations, schema version, start time

Table 4. HDF5 recording schema (version 1).

Files rotate every 15 minutes or 512 MB. During recording, files use the suffix `.partial.h5`; on rotation, the file is atomically renamed to `.h5`. Heartbeat rows are written every 200 ms to ensure the frame index remains current even during quiet periods.

9.2 Playback

The playback service loads an HDF5 file and replays detections through the tracking pipeline. Two modes are available:

Recorded tracks: uses stored track snapshots directly (deterministic replay).

Recompute tracks: feeds stored detections through the live tracking pipeline, enabling comparison of algorithm changes against the same input data.

10 Visualisation

10.1 Dash Web Dashboard

The primary visualisation is a Plotly Dash application serving at `http://localhost:8080/`. Key views:

- **2×2 Range-Doppler grid:** one heatmap per beam with CFAR detection markers and track trails overlaid.
- **Geographic map:** Plotly scattermap (OpenStreetMap tiles) showing projected track positions and ADS-B aircraft.
- **Track table:** active tracks with ID, status, range, Doppler, SNR, and ADS-B correlation info.
- **Metrics panel:** CPU usage, memory, processing latency, frame rate.
- **CFAR controls:** real-time tuning of guard cells, training cells, and P_{fa}.

10.2 Rust Terminal UI

An emerging Rust TUI (`tui/`) built with ratatui receives Range-Doppler mosaics and track data via WebSocket using a TLV (Type-Length-Value) binary protocol. The TUI uses the Kitty graphics protocol for inline image rendering of Range-Doppler heatmaps, providing a lightweight alternative to the full web dashboard for low-power or headless deployments.

11 Configuration and Deployment

11.1 Configuration

All configuration is managed via Pydantic `BaseSettings` models in `config/settings.py`. Settings can be provided via:

1. Environment variables with `PASSIVE_RADAR_` prefix
2. A `.env` file in the project root
3. Constructor arguments in code

Minimal deployment configuration

```
# .env file
PASSIVE_RADAR_PI_IP=192.168.1.50
PASSIVE_RADAR_GEOMETRY__RX_LAT=-34.9285
PASSIVE_RADAR_GEOMETRY__RX_LON=138.6007
PASSIVE_RADAR_GEOMETRY__TX_LAT=-34.8500
PASSIVE_RADAR_GEOMETRY__TX_LON=138.5800
PASSIVE_RADAR_ADSB__ENABLED=true
PASSIVE_RADAR_RECORDING__ENABLED=true
```

Key configuration groups: `ServerConfig` (network), `BeamConfig` (per-beam DSP parameters), `CFARConfig` (detection thresholds), `TrackConfig` (Kalman tuning), `GeometryConfig` (receiver/transmitter locations), `ADSBConfig` (dump1090 endpoint), `RecordingConfig` (HDF5 output).

11.2 Running the System

```
# Full system (auto-discover Pi via mDNS)
uv run python run_server.py

# Manual Pi IP with verbose logging
uv run python run_server.py --pi-ip 192.168.1.50 --verbose

# Headless with synthetic data (no hardware needed)
uv run python run_headless.py --mock --frames 100

# Dashboard only (reads from output/ directory)
uv run python run_dashboard.py
```

11.3 Field Deployment Checklist

1. Connect KrakenSDR to Pi; connect reference + 4 surveillance antennas.
2. Identify a nearby broadcast transmitter; note its frequency and location.
3. Configure receiver and transmitter coordinates in `.env`.
4. Set beam azimuths (degrees from true north) for each surveillance antenna.
5. Start the server; open the dashboard at `http://localhost:8080/`.
6. Verify detections appear on the Range-Doppler display.
7. Enable ADS-B to validate track positions against known aircraft.
8. Enable recording for offline analysis.

12 Performance

Component	Latency	Notes
Network receive (UDP)	<5 ms	Pi → workstation
Wiener clutter cancellation	5–10 ms	Per beam
Range-Doppler generation (CAF)	30–60 ms	Per beam (bottleneck)
CLEAN sidelobe suppression	~20 ms	Per beam (optional)
CA-CFAR detection	2–5 ms	Per beam
Kalman prediction + update	<1 ms	Per beam (~10 tracks)
GNN association (Hungarian)	<1 ms	Per beam
Geographic projection	<1 ms	Per track
ADS-B correlation	1–5 ms	Background, non-blocking
End-to-end	**100–200 ms**	IQ arrival to track display

Table 5. Latency budget. The FFT-based CAF computation dominates processing time.

Typical resource usage on an Apple M1 with 4-beam processing: 70–110% CPU (across 4 workers + main process), 700–950 MB memory. Network bandwidth from the Pi is approximately 1.6 MB/s for 5 channels at 2048 samples/CPI and 80 FPS.

13 Limitations

13.1 Per-Beam Tracking Only

Description: Each beam maintains its own independent track list. The system cannot fuse observations of the same target across multiple beams.

Workaround: An operator can manually correlate tracks across beams using the dashboard. ADS-B correlation implicitly identifies multi-beam targets.

Planned resolution: Cross-beam track fusion is planned for v3.0, requiring multi-beam angle estimation and a joint state estimator.

13.2 Bistatic Range Ambiguity

Description: A single bistatic range measurement maps to an ellipsoid, not a unique point. The beam azimuth constrains the position but with significant angular uncertainty (±2–5°).

Workaround: Use ADS-B correlation to validate and correct position estimates. Accurate transmitter/receiver geometry calibration reduces error.

Planned resolution: Multi-transmitter fusion (using multiple FM stations simultaneously) could provide triangulation. No current implementation plan.

13.3 Manual CFAR Threshold Tuning

Description: The default CFAR parameters work reasonably in typical conditions but require manual adjustment for different environments (urban clutter, rural open sky, different transmitter types).

Workaround: Use the dashboard CFAR controls to adjust parameters in real time and observe the effect on detection counts.

Planned resolution: Automated threshold adaptation based on noise floor statistics is under investigation.

13.4 Reference Signal Quality Dependency

Description: The Wiener filter assumes a clean reference signal. Multipath on the reference antenna or a weak transmitter degrades clutter cancellation and increases the noise floor.

Workaround: Ensure the reference antenna has a clear line-of-sight to the transmitter. Use a directional antenna for the reference channel.

Planned resolution: Adaptive reference signal conditioning is a research topic; no current implementation plan.

13.5 KrakenSDR Hardware Lock-in

Description: The system is designed for the KrakenSDR's 5-channel coherent receiver. Other SDR hardware (HackRF, USRP, BladeRF) would require new driver implementations.

Workaround: The Rust `rtltcp` server (`/Users/simonknight/dev/rtltcp/`) abstracts over RTL-SDR, AirSpy, and AirSpy HF+ via a polymorphic `SDRDevice` trait, but integration with the passive radar pipeline is not yet implemented.

Planned resolution: Out of scope for v2; potential v3 feature.

14 Future Work

GPU acceleration (Phase 22–23): CuPy-based CAF computation for ~10× speedup on CUDA hardware. Stubs exist but are untested.

Rust DSP backend (Phase 24–26): Pure-Rust signal processing pipeline using rustfft, nalgebra, and rayon. Enables native execution on Raspberry Pi and Jetson without a Python runtime. Foundational crate `dsp-core/` is 80% complete.

Cross-beam track fusion (v3): Joint multi-beam tracking with angle estimation and a combined state vector.

Machine learning classification: Real-time target classification using micro-Doppler signatures (propeller vs. rotor vs. fixed-wing).

Multi-transmitter operation: Simultaneous exploitation of multiple FM stations for bistatic triangulation.

References

[1] H. D. Griffiths and C. J. Baker, *An Introduction to Passive Radar*, Artech House, 2017.

[2] T. Peto, "pyAPRiL: Python ADP Passive Radar Library," GitHub, 2020. https://github.com/petotamas/pyAPRiL

[3] KrakenSDR, "KrakenSDR: 5-Channel Coherent RTL-SDR," https://www.krakenrf.com/krakensdr/

[4] R. Labbe, "FilterPy: Kalman filtering and optimal estimation library," GitHub, 2015. https://github.com/rlabbe/filterpy

[5] H. Kuschel, D. Cristallini, and K. E. Olsen, "Tutorial: Passive Radar Tutorial," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 2, 2019.