Multi-Drone Agricultural Monitoring System

Theoretical Framework & Implementation Logic

Development Team

December 12, 2025

Contents

1 Introduction

This document outlines the theoretical foundations and decision-making logic behind the Multi-Drone Agricultural Monitoring System. The system deploys swarms of autonomous drones to monitor farmland, assess drought risks using meteorological data, and validate data integrity through multi-sensor fusion.

2 Theoretical Background

2.1 Drought Probability Model

The system estimates drought risk based on the methodology proposed in *"DroughtCast: A Machine Learning Forecast of the United States Drought Monitor"* (Brust et al., 2021).

2.2 Foundation Research

The project builds upon the swarm architecture detailed in *"Multi-Robot Communication-Aware Cooperative Belief Space Planning with Inconsistent Beliefs: An Action-Consistent Approach"* (Kundu et al., IROS 2024).

Rather than relying on single inputs, the model aggregates multiple meteorological indices:

- **SPI (Standardized Precipitation Index)**: Measures rainfall deficit.
- **SMI (Soil Moisture Index)**: Assesses water availability in the root zone.
- **VCI (Vegetation Condition Index)**: Evaluates crop health via satellite imagery proxies.
- **TCI (Temperature Condition Index)**: Accounts for thermal stress.

The final probability $P_{drought}$ is derived from a weighted non-linear combination of these features, normalized to a $[0, 1]$ range using a logistic function:

$$P(y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + ... + \beta_n X_n)}} \tag{1}$$

2.3 Sensor Fusion (Inverse-Variance Weighting)

To mitigate sensor noise and errors, the system employs **Inverse-Variance Weighting** for merging probabilities from multiple drones. This method gives more weight to precise sensors (low variance) and less weight to noisy ones.

Given a set of measurements x_i with known variances σ_i^2, the fused estimate $\hat{x}$ and its variance $\hat{\sigma}^2$ are calculated as:

$$w_i = \frac{1}{\sigma_i^2} \tag{2}$$

$$\hat{x} = \frac{\sum_i w_i x_i}{\sum_i w_i} \tag{3}$$

$$\hat{\sigma}^2 = \frac{1}{\sum_i w_i} \tag{4}$$

This ensures that the final fused probability is statistically the most reliable estimate possible given the available data.

2.4 Fault Detection Logic

The system automatically detects malfunctioning sensors using statistical hypothesis testing. A sensor reading x_{sensor} is flagged as **faulty** if it deviates significantly from the expected model prediction x_{model} or the consensus of other drones.

The fault condition is triggered if:

$$|x_{sensor} - x_{model}| > (2 \cdot \sigma_{noise} + \delta_{threshold}) \tag{5}$$

Where:

- σ_{noise} is the standard deviation of the sensor (noise floor).
- $\delta_{threshold}$ is a configurable safety margin (e.g., 15%).

If a fault is detected, the system automatically dispatches an **Auditor Drone** from the reserve pool to verify the reading.

2.5 Decentralized Swarm Ranging (INFOCOM 2021)

The implemented localization system operates on the principle of **Relative Positioning** via Ultra-Wideband (UWB) ranging. Unlike GPS, which provides absolute coordinates, UWB modules measure the Time-Of-Flight (ToF) between transceiver pairs to determine distance d_{ij} between Drone i and Drone j. The system resolves the relative swarm geometry using **Multilateration**:

$$\min_{x,y} \sum_{i=1}^{N} (\sqrt{(x - x_i)^2 + (y - y_i)^2} - d_{measured})^2 \tag{6}$$

This optimization problem (Least-Squares) allows the swarm to maintain formation and collision avoidance even in GPS-denied environments.

2.6 Energy-Constrained Coverage (ICRA 2024)

The coverage problem is modeled as a variation of the **Vehicle Routing Problem (VRP)** with energy constraints. The classic coverage objective is to visit all target nodes V such that the total path length is minimized. However, our implementation introduces a battery constraint, $B(t) > B_{critical}$. The system solves this by introducing a mobile recharging station (UGV). The planning logic transforms into a **Rendezvous Problem**: finding the optimal intersection point P_{meet} and time t_{meet} where the drone and UGV paths intersect before $B(t) \to 0$.

2.7 Cooperative Belief Space Planning (IROS 2024)

While the current implementation simplifies the full belief space planning, the theoretical foundation relies on reducing state uncertainty Σ. In a cooperative belief space, the uncertainty of the entire swarm state X is coupled. Effective planning minimizes the trace of the covariance matrix:

$$J = \text{tr}(\Sigma_{swarm}) \tag{7}$$

The system now explicitly computes this metric ($\text{tr}(\Sigma) \approx \text{tr}((J^T J)^{-1} \cdot \text{MSE})$) in real-time within the `swarm_localization` node, providing a live "Belief Uncertainty" score to the console. Our active "Auditor" drone recruitment is a direct application of this principle: deploying an agent specifically to reduce the uncertainty (variance) of a high-risk belief state.

3 Frequently Asked Questions (FAQ)

3.1 Probabilty & Data Merging

Q: How do we merge probabilities from multiple drones?

A: We use a technique called **Sensor Fusion** (specifically, Inverse-Variance Weighting). Imagine Drone A reports a drought risk of 60% with high confidence (low noise), and Drone B reports 40% with low confidence (high noise). Instead of just averaging them to 50%, the system "trusts" Drone A more. The mathematical formula (see Section 2.2) ensures that the final combined probability leans closer to the more accurate sensor.

Q: What happens if the Model and the Sensor disagree?

A: The system trusts the drone's physical sensor reading *unless* the deviation is statistically impossible (see Fault Detection).

- **Small Disagreement**: The system fuses the model prediction and sensor reading mathematically to find a middle ground.
- **Large Disagreement**: The system flags the sensor as potentially faulty and dispatches a second drone (Auditor) to double-check.

3.2 Drone Behavior & Allocation

Q: How are drones assigned to areas?

A: Assignment is based on **Risk Prioritization**. 1. **High Risk Areas ($> 70\%$):** Get multiple drones (up to 3) to ensure frequent monitoring. 2. **Medium Risk Areas ($40-70\%$):** Get at least 1 drone. 3. **Low Risk Areas:** Monitored if resources allow.

We also keep a **Reserve Pool** (approx. 10% of the fleet) idle at the base. These reserves are only launched if a sensor failure is detected and an Auditor is needed immediately.

Q: Why do drones sometimes hover instead of moving?

A: This is usually due to the **Geofencing Safety Protocol**. If a drone drifts near the boundary of its assigned circular area, it enters a "Hold" or "Correction" state. It stops exploring and applies a velocity vector towards the center of the area to ensure it doesn't fly into restricted airspace or crash into a neighboring sector.

4 Implementation Analysis

4.1 Expected vs. Actual Implementation

The initial design specifications (outlined in `expected_work.md` and the reference paper *DroughtCast*) proposed an end-to-end ML pipeline. Below is a comparison of the theoretical design versus the current simulated implementation.

Feature	Expected Design	Current Implementation
Drought Prediction	LSTM Neural Network trained on historical Kaggle meteorological data (SPI, SMI, VCI).	**LSTM Integrated**: The system runs a trained PyTorch LSTM model sequence-to-one inference inside the ROS loop (`drought_probability_model.py`).
Data Sources	Real-time satellite imagery (MODIS) and weather station APIs.	**Historical Data**: Ingests CSV time-series data (Kaggle US Drought Meteorological Data) for 90-day lookback inference.
Coverage Optimization	`scipy.optimize` for perfect non-overlapping circles.	**Greedy Allocation**: Dynamic assignment based on immediate risk priority and proximity (`area_allocation.py`).
Fault Detection	Statistical hypothesis testing (2σ deviation).	**Implemented exactly as designed**. Compares sensor readings against the model's baseline prediction.
Swarm Ranging	Full centralized graph optimization for all N drones.	**Decentralized Multilateration**: Implemented a lightweight Least-Squares solver per drone (`swarm_localization.py`) consuming simulated UWB ranges.
Energy Planning	Complex VRP with Time Windows (VRPTW) and charging constraints.	**Cooperative Rendezvous**: Simplified reactive logic where Critical drones trigger a UGV rendezvous response (`energy_planner.py`).

Table 1: Design vs. Implementation Comparison

4.2 Key Code Implementations

The core logic for our **Sensor Fusion** and **Fault Detection** is critical to the system's reliability. Below are the specific functions driving these behaviors.

4.2.1 Sensor Fusion Logic

We use **Inverse-Variance Weighting** to merge conflicting probability estimates. This ensures that a drone with a noisy sensor (high variance) has less influence on the final decision than a drone with a precise sensor.

```python
def fuse_probabilities(prob_a, var_a, prob_b, var_b):
    """
    Merges two probability estimates using inverse-variance weighting.
    Formula: (w1*x1 + w2*x2) / (w1 + w2) where w = 1/variance
    """
    weight_a = 1.0 / var_a
    weight_b = 1.0 / var_b

    fused_prob = (weight_a * prob_a + weight_b * prob_b) / (weight_a + weight_b)
    fused_var = 1.0 / (weight_a + weight_b)

    return fused_prob, fused_var
```

Listing 1: Inverse-Variance Fusion Logic

4.2.2 Fault Detection Threshold

The system flags a sensor as "Faulty" if its reading deviates from the expected model prediction by more than dynamic threshold (driven by the sensor's known noise profile).

```python
def detect_fault(sensor_val, model_val, sensor_noise_std):
    # Threshold is 2 * standard_deviation (95% confidence interval)
    # plus a small base margin (0.15)
    dynamic_threshold = (2 * sensor_noise_std) + 0.15

    deviation = abs(sensor_val - model_val)

    if deviation > dynamic_threshold:
        return True  # FAULT DETECTED
    return False
```

Listing 2: Adaptive Fault Detection

5 File Structure & Component Overview

This section provides a detailed breakdown of the codebase organization, highlighting the purpose of key directories and files.

5.1 Directory Hierarchy

- `scripts/`: Contains the core Python ROS nodes.
  - `area_explorer.py`: The main mission controller. Orchestrates risk analysis, allocation, and drone movement.
  - `area_allocation.py`: Implements the logic for Phase 2-4 drone allocation (High-/Med/Low risk assignment).
  - `drought_probability_model.py`: Wrapper for the LSTM inference engine.
  - `swarm_localization.py`: (Research) Decentralized UWB-based positioning logic.
  - `energy_planner.py`: (Research) Battery monitoring and UGV rendezvous coordination.

  - `uwb_simulator.py`: Simulates hardware UWB ranges.
  - `ugv_manager.py`: Controls the Mobile Charging Station.
- `launch/`: ROS launch files for system startup.
  - `explore_areas.launch`: The master entry point. Launches the simulation, spawns drones, and starts the explorer node.
  - `spawn_drones.launch`: Helper script to spawn 18 quadcopters in Gazebo with specific namespaces (`drone_0` to `drone_17`).
- `config/`: Configuration files.
  - `areas.yaml`: Defines the 10 monitored farmland areas (coordinates, crop type, historical drought data).
- `LSTM/`: Deep Learning resources.
  - `model.py`: PyTorch definition of the `DroughtLSTM` class (Input: 6 features, Hidden: 64 units).
  - `lstm_model.pth`: Pre-trained binary weights for the model.
- `worlds/`: Simulation environments.
  - `field_areas.world`: A Gazebo world file pre-configured with the farmland terrain.
- `models/`: 3D assets.
  - `quadcopter/`: SDF model description for the Iris drone used in the simulation.

6 Codebase Architecture & Logic

This section details the functional implementation of the active nodes, mapping specific classes and functions to the theoretical research papers.

6.1 DroughtCast Implementation (Brust et al., 2021)

Core Logic: `drought_probability_model.py`

The system implements the multi-index probabilistic model proposed in *DroughtCast* using a Long Short-Term Memory (LSTM) network.

- **Class**: `DroughtLSTM` (in `LSTM/model.py`)
- **Role**: The neural architecture. Defined as a 2-layer LSTM with 64 hidden units, followed by a fully connected layer and Sigmoid activation.
- **Why?**: Captures temporal dependencies in meteorological time-series data (90-day look-back).
- **Function**: `predict_from_csv(self, csv_path)`
- **How it works**:
  1. Loads 90 days of daily stats (SPI, SMI, VCI, etc.) from CSV.
  2. Normalizes data to $[0, 1]$.
  3. Feeds tensor of shape $(1, 90, 6)$ into the model.
  4. Returns $P(drought) \in [0, 1]$.

6.2 Risk-Aware Allocator (Logic Layer)

Core Logic: `area_allocation.py`
This module translates the abstract probabilities into concrete drone assignments.

- **Class**: `DynamicDroneAllocator`
- **Function**: `allocate_drones(areas, drones)`
- **Mechanism**:
  - **Sorting**: `areas.sort(key=lambda x: x.drought_probability, reverse=True)`.
  - **Phase 1 (Baseline)**: `assign_drone(area_id)` for all areas to ensure minimal coverage.
  - **Phase 3 (High-Risk Injection)**: For areas with $P > 0.7$, iterate and assign additional drones up to `max_drones_per_area`.
  - **Reserves**: Uses `reserve_percentage` to keep a portion of the fleet idle for fault-recovery (Auditor missions).

6.3 Main Controller (ROS Integration)

Core Logic: `area_explorer.py`
The `area_explorer` node binds the logic together into a ROS application.

- **Class**: `DroneExplorer`
- **Role**: Manages the state machine for a single drone (Init $\rightarrow$ Explore $\rightarrow$ Return).
- **Function**: `explore()`
- **Key Implementation**:
  - **Path Planning**: Generates "Lawnmower" grid waypoints inside `is_within_area_bounds()`.
  - **Flight Control**: Implements a P-Controller.
  - **Critical Fix**: The raw velocity vector (v_x, v_y) is rotated by the drone's Yaw ψ to ensure correct movement relative to the drone's heading:

$$R(\psi) = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \tag{8}$$

6.4 Swarm Ranging (INFOCOM 2021)

Core Logic: `swarm_localization.py`
Implements the "Range-Based Relative Positioning" protocol.

- **Class**: `SwarmLocalization`
- **How**: Subscribes to `/swarm/uwb_ranges` (mocked hardware).
- **Function**: `compute_position()`
- **Algorithm**: Non-linear Least Squares.

$$\text{minimize } \sum (d_{measured} - ||P_{est} - P_{anchor}||)^2 \tag{9}$$

- **Role**: Allows drones to know where they are relative to neighbors without GPS.

6.5 Cooperative Belief Space (IROS 2024)

Core Logic: Integrated into `swarm_localization.py`

- **Feature**: Real-time uncertainty quantification.
- **Implementation**: After solving for position P, we calculate the Jacobian J.
- **Metric**: Trace of the Covariance Matrix ($\text{tr}(\Sigma)$), where $\Sigma \approx (J^T J)^{-1}$.
- **Role**: Provides a "Confidence Score" for the location estimate, used to trigger auditor drones if uncertainty is too high.

6.6 Energy-Constrained Coverage (ICRA 2024)

Core Logic: `energy_planner.py`

- **Class**: `EnergyAwarePlanner`
- **Role**: Fleet resource manager.
- **How**:
  1. **Monitoring**: `bat_cb()` updates internal state dict.
  2. **Thresholding**: `if battery < 30.0`: Mark as Critical.
  3. **Action**: `coordinate_rendezvous()` commands the UGV (Mobile Charger) to the drone's last known odometry position.

7 Future Work & Limitations

7.1 Current Limitations

- **Simulated Weather:** The system currently does not ingest live weather API data; it relies on a sophisticated probability generator.
- **2D Movement:** Drones currently operate at a fixed altitude. True 3D terrain following is not yet implemented.
- **Simplified Physics:** Battery consumption is modeled as a linear drain, ignoring wind resistance and varied payload mass.

7.2 Research Modules Implemented

- **Swarm Ranging Protocol** (INFOCOM 2021): Implemented decentralized UWB-based localization. The `swarm_localization` node uses multilateration to estimate relative positions without relying on global GPS.
- **Energy-Aware Path Planning** (ICRA 2024): Integrated a battery discharge model and a mobile charging station (`ugv_manager`). The `energy_planner` coordinates rendezvous events for drones with critical battery levels (<30%).

8 Implementation FAQs

Q: How was the INFOCOM 2021 Swarm Ranging paper implemented?
A: We simulated the hardware layer by creating a `uwb_simulator` node that calculates ground-truth distances and adds Gaussian noise to mimic real Ultra-Wideband sensors. A decentralized `swarm_localization` node then consumes these noisy ranges and uses a Multilateration (Least Squares) algorithm to estimate relative positions. This validates the paper's core proposition of achieving relative localization without global GPS.

Q: How was the ICRA 2024 Energy Planning paper implemented?
A: We abstracted the paper's energy constraints into a `Battery` class within the drone controller, which models discharge rates based on flight state (hovering vs. flight). The simplified "coverage with mobile recharging" logic is handled by a `energy_planner` node, which identifies critical drones and commands a simulated UGV (Mobile Charging Station) to rendezvous with them, effectively implementing the cooperative coverage-charging cycle.

Q: What inferences were made from the research papers?
A: We inferred that for a ROS-based simulation, the *behavioral logic* (rendezvous, range-based fixing) was more critical than exact physical modelling. Thus, we used linear approximations for battery drain and UWB noise models. We also adapted the centralized optimization algorithms from the papers into reactive ROS nodes to suit real-time simulation constraints.

8.1 Future Roadmap

1. **Live Weather APIs**: Replace the CSV data loader with real-time fetchers for OpenWeatherMap/MODIS APIs.

2. **Hardware Deployment:** Port the ROS nodes to a swarm of Bitcraze Crazyflie 2.1 drones for real-world field testing.