📄 Phase 7.3 – Statistical Anomaly Detection (Hybrid Approach)

7.3.1 Purpose of this Phase

Up to now the system on the **Arduino UNO Q** can:

- Listen to the CAN bus
- Decode ECU voltages via the `harness_demo.dbc`
- Apply **rule-based harness fault detection** in real time
- Raise clear alerts for:
  - Harness A (ECU A only low)
  - Harness B (ECU B only low)
  - Harness C (both ECUs low vs DCDC)

Phase 7.3 adds a **statistical layer on top** of those rules to:

1. Detect **early, subtle drift** before rule thresholds are hit
2. Characterise **trends over time** (which direction and how fast)
3. Lay the groundwork for simple **Remaining Useful Life (RUL-style) reasoning**, without needing heavy ML or cloud compute

This is **not** a full prognostics stack. It's a **hybrid design**: lightweight analytics on the UNO Q, with a clear path to more advanced methods later.

7.3.2 Signals and Derived Quantities

The statistical layer works on the **same decoded signals** as the rules:

- `V_DCDC` (DCDC_Output_Voltage)
- `V_A` (ECUA_Supply_Voltage)
- `V_B` (ECUB_Supply_Voltage)

We reuse the deltas:

- ΔA = V_DCDC − V_A
- ΔB = V_DCDC − V_B

From these we derive:

- **Level metrics** – current ΔA, ΔB
- **Smoothed metrics** – filtered versions of ΔA, ΔB
- **Trend metrics** – rate-of-change of ΔA, ΔB

These are all cheap to compute and suitable for the UNO Q's Linux environment.

7.3.3 Lightweight Features on the UNO Q

We implement three simple statistical tools:

(1) Sliding window mean

Over the last **N samples** (e.g. N = 50 at 100 ms ≈ 5 seconds):

- `mean_ΔA` = average of ΔA over the window
- `mean_ΔB` = average of ΔB over the window

Purpose:

- Filters out short spikes
- Gives a "local baseline" around which we can measure drift

(2) Exponential smoothing (EWMA)

We also maintain an **exponential moving average**:

- `ewma_ΔA ← α * ΔA_now + (1 - α) * ewma_ΔA_prev`
- `ewma_ΔB ← α * ΔB_now + (1 - α) * ewma_ΔB_prev`

Where α is a smoothing factor, e.g.:

- α ≈ 0.05 → strong smoothing, slow response
- α ≈ 0.2 → faster response, less smoothing

Purpose:

- Stable, low-memory estimate of "underlying" delta
- Robust against noise, still responsive to gradual drift

(3) Approximate slope (trend)

We estimate a **trend** (rate-of-change) over the window:

```
trend_ΔA ≈ (ΔA_latest - ΔA_oldest_in_window) / window_duration
trend_ΔB ≈ (ΔB_latest - ΔB_oldest_in_window) / window_duration
```

For example, if ΔA grows from 0.2 V to 0.8 V over 60 seconds:

- trend_ΔA ≈ (0.8 − 0.2) / 60 ≈ 0.01 V/s

Purpose:

- Distinguishes:
  - **Sudden jumps** (bad connection, intermittent contact)
  - **Slow monotonic drift** (classic corrosion / resistance growth)

7.3.4 Statistical Anomaly Rules

The **statistical layer does not replace the hard rules**; it augments them.

We define **early warning conditions** such as:

1. Drift warning (Harness A example)

> "ΔA is still below the hard alarm threshold but is **steadily trending up**."

Concretely:

- `ewma_ΔA` > 0.3 V
- `trend_ΔA` > 0.002 V/s
- Duration condition: above for ≥ 30 seconds

This is logged as:

```
[EARLY][HARNESS_A_DRIFT] ΔA increasing steadily (ewma=...,
trend=...)
```

2. Volatility anomaly

If the short-term standard deviation of ΔA or ΔB grows:

- `std(ΔA_window)` > σ_threshold

Interpretation:

- Contact is becoming intermittent
- Loose terminal / vibration / micro-arcing

This can be flagged:

```
[EARLY][HARNESS_A_NOISY] ΔA variance high (std=...)
```

3. Multi-node statistical confirmation

We ensure that:

- For Harness A drift:
    - ΔA trending up
    - ΔB stable
    - DCDC reasonably stable

So we require, for example:

- `trend_ΔA > drift_min`
- `|trend_ΔB| < drift_tolerance`

- `|trend_V_DCDC| < small_tolerance`

This **reinforces localization**: we only call it a harness issue if the pattern is consistent with the physical model.

7.3.5 Simple RUL-Style Estimation (Conceptual)

While not a full RUL engine, we can derive a **rough Remaining Useful Life style metric** from:

- Current delta: Δ
- Trend: trend_Δ

If we know:

- Soft limit: `Δ_warn` (e.g. 0.5 V)
- Hard limit: `Δ_fail` (e.g. 1.5 V)

When trend_Δ > 0 (degradation worsening), we can approximate:

```
time_to_warn ≈ max(0, (Δ_warn - current_Δ) / trend_Δ)
time_to_fail ≈ max(0, (Δ_fail - current_Δ) / trend_Δ)
```

We then present it to the logs as **qualitative RUL**:

- `> 60 min` → "Low risk"
- `10–60 min` → "Plan maintenance"
- `< 10 min` → "High risk – intervention soon"

This is intentionally **coarse**, but it gives a taste of:

> "Not only do we know you have a problem, we have an idea of how fast it is getting worse."

We **do not** hard-wire this into safety decisions; it is an **advisory metric**.

7.3.6 Integration with the Existing Rule Engine

The runtime now has **three layers**:

1. **Raw decode layer**
   - From CAN frames → ECU voltages via DBC
   - Already implemented and validated
2. **Rule-based fault layer**
   - Hard logical conditions
   - Distinguishes Harness A/B/C
   - emits `[ALERT][HARNESS_X] ...`
3. **Statistical layer (this phase)**
   - Monitors drift, volatility, trend
   - Issues early warnings and RUL-style estimates
   - emits `[EARLY][HARNESS_X_DRIFT] ...` and optionally RUL text

The **key design decision**:

> Rules always remain the **primary functional diagnostic**,
> statistics act as an **early warning and context provider**.

This keeps the system explainable and robust.

7.3.7 Resource Constraints and UNO Q Considerations

Our design is conscious of the UNO Q environment:

- **CPU:** simple arithmetic, no heavy matrix operations
- **RAM:** only small sliding windows (e.g. 50–100 samples) per signal
- **Storage:** all statistics computed in real-time; nothing stored long-term unless we choose to log them

We avoid:

- Large numpy/scipy stacks
- Heavy ML frameworks
- Compiled C extensions for this phase

This keeps deployment and debugging on the UNO Q straightforward.

7.3.8 Limitations and Future Extension Path

Limitations of the current approach:

- Thresholds and windows are **hand-tuned**, not data-driven
- RUL estimates are **linear extrapolations**, not physically derived models
- Environmental effects (temperature, load) are not yet modeled

Future extensions (if project extends beyond current scope):

- Use real fleet data to:
  - learn baseline distributions of ΔA, ΔB
  - calibrate trend thresholds automatically

- Introduce simple **Bayesian** or **Kalman** filters for better noise handling
- Add **temperature and load** as contextual features
- Move heavy training to the cloud, keep only lightweight inference on the UNO Q

7.3.9 Summary and Link to Next Phases

By the end of **Phase 7.3**, the system can:

- Detect harness issues through **deterministic rules** (Phase 7.2)
- Monitor **drift and volatility** statistically in real time
- Provide **early warnings** and **rough RUL-style** indications
- Still run entirely on the **UNO Q** with modest resource usage

Next Step: Phase 7.4 – Edge vs Off-Board Trade-Offs

In Phase 7.4 we will document:

- Which logic must run **on the UNO Q** (safety, latency)
- Which analytics can be **offloaded** (detailed RUL, model refinement)
- What data to log and send off-board (CAN snippets, deltas, trends, alerts)
- How this balances:
    - bandwidth
    - cost
    - updatability
    - cybersecurity