OPS243-C Radar Data Pattern Analysis

Data Pattern Analysis from Your Sample

Vehicle Detection Event Sequence

Looking at your sample data, I can identify a clear vehicle detection pattern:

Phase 1: Background/Idle State (104.4-105.1s)

```
json
```

```
{"time": "104.438", "unit": "m", "magnitude": "10874", "range": "2.1"}
{"time": "104.506", "unit": "m", "magnitude": "10880", "range": "2.1"}
```

- **High baseline magnitude**: ~10,800-10,900 (background reflection)
- **Stable range**: 2.1m (consistent background)
- **No speed readings**: Vehicle not yet detected

Phase 2: Vehicle Approach Detection (105.1-105.5s)

```
json
```

```
{"time": "105.110", "unit": "m", "magnitude": "9325", "range": "2.1"}
{"time": "105.122", "unit": "mps", "magnitude": "23", "speed": "1.0"}
{"time": "105.178", "unit": "m", "magnitude": "5078", "range": "2.1"}
{"time": "105.190", "unit": "mps", "magnitude": "683", "speed": "1.3"}
```

- **Magnitude drops**: From 10,874 → 9,325 → 5,078 (vehicle entering detection zone)
- **Speed measurements begin**: First speed reading of 1.0 m/s (approaching)
- **Range changes**: Some readings show 0.7m (closer vehicle detection)

Phase 3: Vehicle Passing/Peak Detection (105.5-106.0s)

```
json
```

```
{"time": "105.395", "unit": "mps", "magnitude": "357", "speed": "-1.1"}
{"time": "105.463", "unit": "mps", "magnitude": "148", "speed": "-1.5"}
{"time": "105.869", "unit": "mps", "magnitude": "365", "speed": "-1.7"}
```

- **Speed direction change**: Positive → Negative (vehicle passing sensor)

- **Magnitude fluctuations**: Varying reflection strength as vehicle moves
- **Peak negative speeds**: Up to -1.7 m/s (receding)

Phase 4: Vehicle Departure (106.0-107.8s)

```
json
```


```
{"time": "106.206", "unit": "mps", "magnitude": "512", "speed": "2.2"}
{"time": "106.274", "unit": "mps", "magnitude": "310", "speed": "1.3"}
{"time": "106.464", "unit": "m", "magnitude": "9376", "range": "2.1"}
```


- **Final speed burst**: 2.2 m/s (highest speed reading)
- **Magnitude recovery**: Gradual return to baseline ~10,800
- **Speed readings cease**: Vehicle exits detection zone

Phase 5: Return to Baseline (107.8s+)

```
json
```


```
{"time": "107.812", "unit": "m", "magnitude": "10825", "range": "2.1"}
{"PowerMode": "Idle or Pulse"}
```


- **Stable baseline**: Back to ~10,800 magnitude
- **System idle**: Sensor returns to idle mode

Key Data Characteristics

Magnitude Patterns

Phase	Magnitude Range	Meaning
Background	10,800-10,900	No vehicle present
Detection	2,000-9,000	Vehicle in detection zone
Peak	1,000-3,000	Vehicle closest to sensor
Recovery	5,000-10,800	Vehicle departing

Speed Patterns

Speed Range	Direction	Interpretation
+0.5 to +2.5 m/s	Approaching	Vehicle moving toward sensor
-0.5 to -2.0 m/s	Receding	Vehicle moving away from sensor
±0.3 m/s	Stationary	Vehicle stopped or very slow

Range Behavior

- **2.1m**: Default/background range reading
- **0.7-1.4m**: Vehicle detected at closer range
- **Fluctuating ranges**: Multiple reflection points from vehicle

---

Data Processing Algorithms Needed

1. Vehicle Event Detection Algorithm

python

```python
def detect_vehicle_event(magnitude_history):
    # Threshold-based detection
    baseline = calculate_baseline(magnitude_history[-100:])  # ~10,800
    current_magnitude = magnitude_history[-1]

    # Vehicle detected when magnitude drops significantly
    if baseline - current_magnitude > 2000:  # 2000 magnitude drop
        return True
    return False
```

2. Speed Correlation Algorithm

python

```python
def correlate_speed_with_detection(detection_time, speed_readings):
    # Find speed readings within ±2 seconds of detection
    relevant_speeds = [
        speed for speed in speed_readings
        if abs(speed.timestamp - detection_time) < 2.0
    ]

    # Filter out noise (speeds < 0.3 m/s)
    valid_speeds = [s.speed for s in relevant_speeds if abs(s.speed) > 0.3]

    return valid_speeds
```

3. Direction Classification

python

```python
def classify_direction(speed_sequence):
    positive_speeds = [s for s in speed_sequence if s > 0.3]
    negative_speeds = [s for s in speed_sequence if s < -0.3]

    if positive_speeds and negative_speeds:
        return "passing"  # Vehicle passed by sensor
    elif positive_speeds:
        return "approaching"
    elif negative_speeds:
        return "receding"
    else:
        return "stationary"
```

4. Data Quality Filters

Magnitude Smoothing (Gaussian Filter)

python

```python
import scipy.signal
smoothed_magnitudes = scipy.signal.gaussian_filter1d(raw_magnitudes, sigma=2.0)
```

Speed Outlier Removal (Z-Score Method)

python

```python
import numpy as np
from scipy.stats import zscore

z_scores = np.abs(zscore(speed_readings))
clean_speeds = [speed for speed, z in zip(speed_readings, z_scores) if z < 3.0]
```

Temporal Consistency Check

python

```python
def validate_temporal_consistency(readings):
    # Check for reasonable time intervals between readings
    time_diffs = [readings[i+1].time - readings[i].time for i in range(len(readings)-1)]
    avg_interval = np.mean(time_diffs)

    # Reject readings with unusual timing
    return 0.05 < avg_interval < 0.2  # 50ms to 200ms intervals
```

Real-World Calibration Needs

Environmental Factors

- **Temperature compensation**: Radar frequency drift with temperature
- **Humidity effects**: Signal attenuation in high humidity
- **Rain/snow impact**: Precipitation causes false readings

Installation Factors

- **Height above road**: Affects detection range and accuracy
- **Angle of sensor**: Optimal angle for speed measurement
- **Background reflections**: Buildings, signs, barriers affecting baseline

Vehicle Characteristics

- **Large vehicles**: Higher magnitude readings (trucks, buses)
- **Small vehicles**: Lower magnitude readings (motorcycles, compact cars)
- **Vehicle materials**: Metal vs. composite body effects

Integration with Camera Data

Temporal Synchronization

python

```python
def synchronize_radar_camera(radar_detection, camera_detections):
    # Find camera detections within ±1 second of radar detection
    time_window = 1.0

    synchronized_detections = []
    for cam_detection in camera_detections:
        if abs(cam_detection.timestamp - radar_detection.timestamp) < time_window:
            synchronized_detections.append({
                'camera': cam_detection,
                'radar': radar_detection,
                'confidence': calculate_correlation_confidence(cam_detection, radar_detection)
            })

    return synchronized_detections
```

Cross-Validation

- **Speed validation**: Camera-based speed vs. radar speed
- **Vehicle count validation**: Visual count vs. radar detections
- **Direction validation**: Visual direction vs. Doppler direction

This analysis shows your radar data contains rich information for accurate vehicle detection and speed measurement, with clear patterns that can be reliably processed using the algorithms I've provided!