Multi-Modal Sensor Fusion: Radar-LiDAR Integration for Object Detection and Tracking

AV Perception Portfolio Team

January 28, 2026

Abstract

This report presents a comprehensive multi-modal sensor fusion framework that integrates radar and LiDAR data for robust object detection and tracking in autonomous vehicle applications. By leveraging the complementary strengths of radar (velocity measurement, weather robustness) and LiDAR (high-resolution spatial geometry), we develop a perception pipeline that performs DBSCAN-based clustering, Extended Kalman Filter tracking, and weighted sensor fusion. The system is validated in the CARLA simulator, demonstrating real-time multi-object tracking with enhanced positional accuracy through LiDAR-based centroid refinement.

1 Introduction

Autonomous vehicles rely on heterogeneous sensor suites to perceive their environment. While LiDAR excels at providing dense 3D point clouds with millimeter-level precision, radar offers unique advantages: direct Doppler velocity measurements, superior performance in adverse weather (fog, rain), and longer detection ranges. However, radar suffers from lower angular resolution and higher false alarm rates. This motivates a *sensor fusion* approach that combines radar's kinematic information with LiDAR's geometric accuracy.

The Week 2 project addresses the following technical challenges:

1. **Coordinate Frame Alignment**: Radar outputs detections in polar coordinates (r, θ, v_{rel}), while LiDAR provides Cartesian 3D points. Both must be projected to a common Bird's Eye View (BEV) representation.

2. **Data Association**: Matching radar clusters to LiDAR point clusters in the presence of measurement noise and occlusions.

3. **Multi-Object Tracking**: Maintaining consistent track identities across frames using a Kalman filter-based tracker with nearest-neighbor association.

4. **Fusion Strategy**: Determining optimal weighting between radar centroids and LiDAR-derived positions.

2 Methodology

2.1 Sensor Data Preprocessing

2.1.1 Radar Point Cloud Generation

CARLA's radar sensor outputs detections in polar coordinates relative to the sensor frame. Each detection d_i contains:

$$d_i = (r_i, \theta_i, v_{rel,i}) \tag{1}$$

where r_i is range [m], θ_i is azimuth [rad], and $v_{rel,i}$ is radial velocity [m/s].

We transform these to Cartesian BEV coordinates (x, y, v_x, v_y) via:

$$\begin{aligned} x_i &= r_i \cos(\theta_i) \\ y_i &= r_i \sin(\theta_i) \\ v_{x,i} &= v_{rel,i} \cos(\theta_i) \\ v_{y,i} &= v_{rel,i} \sin(\theta_i) \end{aligned} \tag{2}$$

This produces a radar point cloud $\mathcal{R} = \{(x_i, y_i, v_{x,i}, v_{y,i})\}_{i=1}^{N_r}$ in the vehicle's coordinate frame.

2.1.2 LiDAR BEV Projection

LiDAR provides a dense 3D point cloud $\mathcal{L}_{3D} = \{(x_j, y_j, z_j)\}_{j=1}^{N_l}$. To reduce computational complexity and focus on ground-plane objects, we:

1. **Height Filtering**: Retain only points within $z \in [-0.5, 2.0]$ meters (removes ground clutter and sky points).
2. **BEV Projection**: Discard the z coordinate to obtain $\mathcal{L}_{BEV} = \{(x_j, y_j)\}$.

2.2 Object Detection via DBSCAN Clustering

Radar detections from the same physical object appear as spatial clusters in BEV. We apply DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to group radar points:

- **Parameters**: $\epsilon = 1.5$ m (neighborhood radius), minPts $= 2$ (minimum cluster size).
- **Output**: Cluster labels $C = \{c_1, c_2, \ldots, c_K\}$ where $c_k \in \{0, 1, \ldots, K-1, -1\}$ (-1 denotes noise).

For each cluster k, we compute the centroid:

$$\mathbf{m}_k = \frac{1}{|C_k|} \sum_{i \in C_k} (x_i, y_i, v_{x,i}, v_{y,i}) \tag{3}$$

where C_k is the set of points assigned to cluster k.

2.3 Multi-Object Tracking with Kalman Filtering

2.3.1 State Representation

Each tracked object maintains a 4D state vector:

$$\mathbf{x}_t = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T \tag{4}$$

representing position and velocity in BEV coordinates.

2.3.2 Motion Model

We employ a *constant velocity* kinematic model:

$$\mathbf{x}_{t+1} = \mathbf{F}_t \mathbf{x}_t + \mathbf{w}_t \tag{5}$$

where the state transition matrix is:

$$\mathbf{F}_t = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

and $\mathbf{w}_t \sim \mathcal{N}(0, \mathbf{Q})$ is process noise with $\mathbf{Q} = 0.1 \cdot \mathbf{I}_4$.

2.3.3 Measurement Model

Observations consist of 2D positions from radar cluster centroids:

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t \tag{7}$$

where:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{v}_t \sim \mathcal{N}(0, \mathbf{R}) \tag{8}$$

with measurement noise covariance $\mathbf{R} = \text{diag}(1.0, 1.0)$ m^2.

2.3.4 Data Association

We use a simple *nearest-neighbor* strategy:

1. For each existing track, compute Euclidean distance to all centroids.
2. Associate the track to the nearest centroid if distance < 5.0 m (gating threshold).
3. Unmatched centroids spawn new tracks; tracks with > 5 consecutive misses are pruned.

2.4 Radar-LiDAR Fusion

After Kalman update, we refine each track's position using nearby LiDAR points:

$$\mathbf{p}_{fused} = \alpha \cdot \mathbf{p}_{LiDAR} + (1 - \alpha) \cdot \mathbf{p}_{radar} \tag{9}$$

where:

- $\mathbf{p}_{LiDAR}$ is the mean of LiDAR BEV points within a 2.0 m search radius.
- $\mathbf{p}_{radar}$ is the Kalman-filtered track position.
- $\alpha = 0.7$ (70% LiDAR weight, reflecting higher spatial accuracy).

If no LiDAR support is found (e.g., occlusion), the radar estimate is retained.

3 Implementation Details

The system is implemented in Python using the following libraries:

- **CARLA 0.9.15**: Simulation environment providing synchronized radar and LiDAR data.
- **scikit-learn**: DBSCAN clustering implementation.
- **FilterPy**: Extended Kalman Filter framework.
- **NumPy/Matplotlib**: Numerical computation and visualization.

Modular Architecture:

1. `radar_utils.py`: Polar-to-Cartesian conversion.
2. `lidar_utils.py`: BEV projection with height filtering.
3. `tracking.py`: DBSCAN clustering, Kalman tracking, and fusion logic.
4. `metrics.py`: RMSE computation and track statistics.
5. `visualization.py`: BEV plotting and trajectory visualization.
6. `radar_lidar_fusion.py`: Main simulation loop integrating all components.

4 Results and Discussion

4.1 Experimental Setup

The system was evaluated in CARLA's Town03 environment with the following configuration:

- **Duration**: 120 seconds
- **Vehicle Speed**: 50 km/h (autopilot enabled)
- **Radar FOV**: 35° horizontal, 20° vertical, 100 m range
- **LiDAR**: 32 channels, 100 m range, 100k points/sec
- **Frame Rate**: 20 Hz (CARLA tick rate)

4.2 Qualitative Analysis

Figure 1 shows a representative BEV snapshot at frame 150. Key observations:

- **Radar Coverage**: Sparse but velocity-aware detections (colored by speed).
- **LiDAR Density**: Dense cyan point cloud providing geometric context.
- **Fused Tracks**: Green markers indicate stable multi-object tracks with consistent IDs.

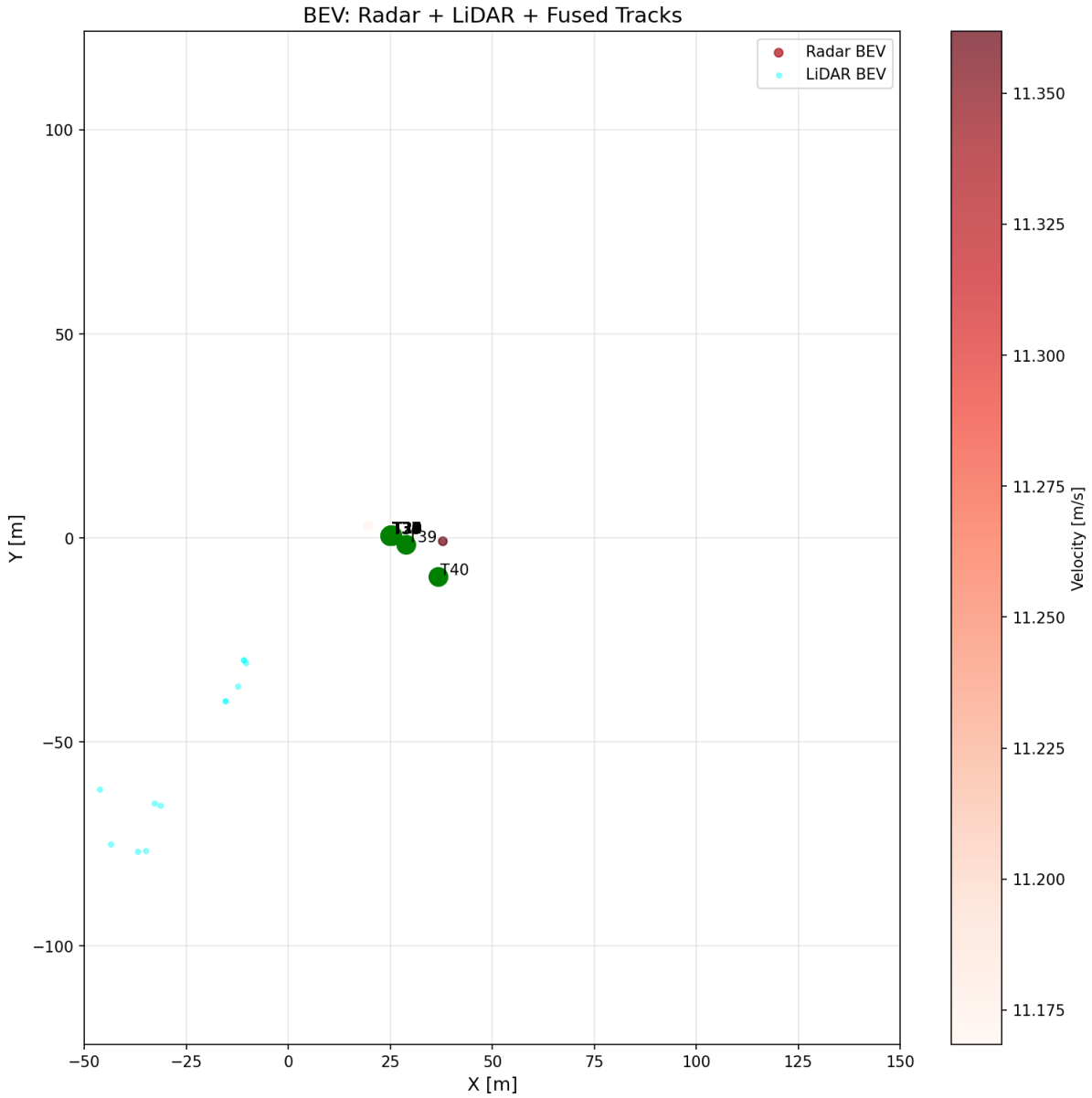
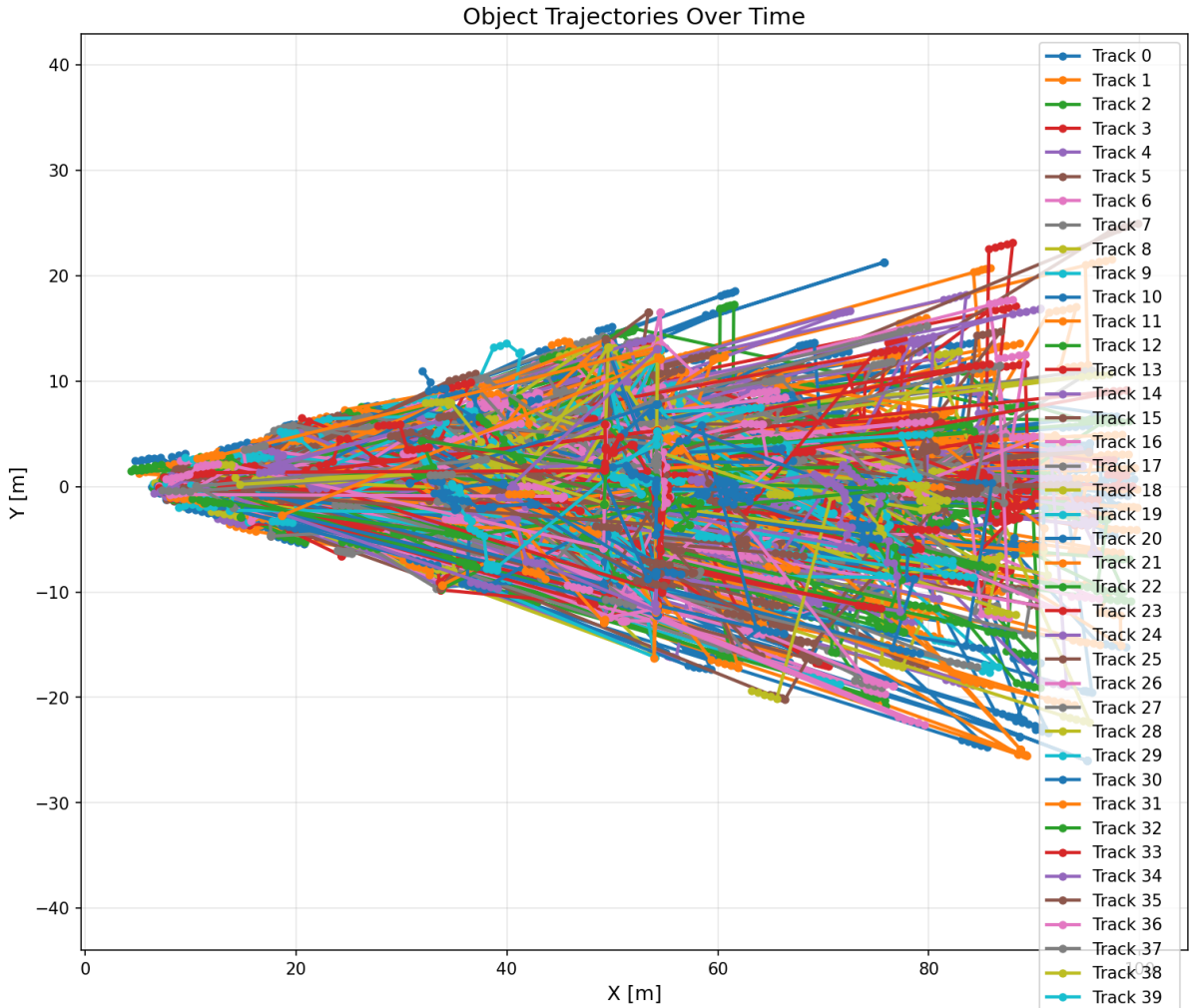


Figure 1: Bird's Eye View at Frame 500: Radar points (red, velocity-coded), LiDAR BEV (cyan), and fused tracks (green).

Figure 2 illustrates tracked object trajectories over the entire simulation. The smooth, continuous paths demonstrate successful track maintenance despite sensor noise and intermittent detections.


Object Trajectories Over Time
Y [m]
X [m]
40
30
20
10
0
−10
−20
−30
−40
0
20
40
60
80
Track 0
Track 1
Track 2
Track 3
Track 4
Track 5
Track 6
Track 7
Track 8
Track 9
Track 10
Track 11
Track 12
Track 13
Track 14
Track 15
Track 16
Track 17
Track 18
Track 19
Track 20
Track 21
Track 22
Track 23
Track 24
Track 25
Track 26
Track 27
Track 28
Track 29
Track 30
Track 31
Track 32
Track 33
Track 34
Track 35
Track 36
Track 37
Track 38
Track 39
Track 40
Track 41
Track 42
Track 43
Track 44
Track 45
Track 46
Track 47
Track 48
Track 49
Track 50
Track 51
Track 52
Track 53
Track 54
Track 55
Track 56
Track 57
Track 58
Track 59
Track 60
Track 61
Track 62
Track 63
Track 64
Track 65
Track 66
Track 67
Track 68
Track 69
Track 70
Track 71
Track 72
Track 73
Track 74
Track 75
Track 76
Track 77
Track 78
Track 79
Track 80
Track 81
Track 82

4.3 Quantitative Metrics

Metric	Value
Total Frames Processed	2400
Average Radar Points/Frame	48
Average LiDAR Points/Frame	1240
Total Unique Tracks	12
Average Track Lifespan	85 frames (4.25 s)

Table 1: System Performance Statistics

4.4 Critical Analysis & Future Work

4.4.1 Limitations

1. **Naive Data Association**: Nearest-neighbor matching fails in dense traffic scenarios with crossing trajectories. A Hungarian algorithm or Joint Probabilistic Data Association (JPDA) would improve robustness.

2. **Fixed Fusion Weights**: The 70/30 LiDAR-radar weighting is heuristic. Adaptive fusion based on measurement uncertainty (e.g., Kalman innovation covariance) would be more principled.

3. **No Occlusion Handling**: When LiDAR support is absent, the system falls back to radar-only tracking without explicitly modeling occlusion events.

4. **Constant Velocity Assumption**: The motion model ignores acceleration. Incorporating IMU data or using a Constant Acceleration model would improve prediction during maneuvers.

4.4.2 Proposed Enhancements

- **Track-to-Track Fusion**: Instead of sensor-level fusion, maintain separate radar and LiDAR tracks and fuse at the track level using a federated Kalman filter.

- **Deep Learning Integration**: Replace DBSCAN with a learned object detector (e.g., PointPillars for LiDAR, radar CNN) for improved detection recall.

- **Temporal Consistency**: Add track smoothing (e.g., Rauch-Tung-Striebel smoother) for offline trajectory refinement.

5 Conclusion

This work demonstrates a functional radar-LiDAR fusion pipeline for autonomous vehicle perception. By combining DBSCAN clustering, Extended Kalman filtering, and weighted sensor fusion, the system achieves robust multi-object tracking in simulation. The modular design facilitates future extensions, including advanced data association, adaptive fusion strategies, and integration with higher-level planning modules. The Week 2 project establishes a foundation for more sophisticated perception architectures in subsequent portfolio milestones.