Radar odometry using mmWave technology for SLAM applications.

Leveraging mmWave Radar Sensor Technology for odometry estimation.

Luis Fernando Rodriguez Gutierrez
Fachhochschule Dortmund
M.Eng. Embedded Systems Engineering
luis.rodriguez001@stud.fh-dortmund.de

***Abstract*—The employment and integration of radar technologies for the implementation of odometry has recently emerged as a promissing alternative to traditional methods, which often rely on visual or LiDAR-based systems. Radar Technology is particularly advantageous due to its robustness in various environmental conditions, such as fog, rain, and dust, where other systems results may degrade.**

This work focus in the estimation of the vehicle's ego-motion using a dual mmWave radar sensor system and an inertial motion sensor.

The proposed pipeline incorporates clustering techniques, point-to-point iterative closest point (ICP) optimization, and Doppler velocity augmentation to address the inherent challenges of sparse and noisy radar point clouds. Notably, the point-to-point ICP approach is advantageous for radar-based odometry as it does not require an initial guess of the transformation, which is often difficult to obtain due to data sparsity and noise. Furthermore, submap aggregation is employed to enhance registration stability across consecutive scans. Experimental evaluations demonstrate that the proposed framework enables consistent ego-motion estimation and highlights the potential of mmWave radar as a cost-efficient solution for autonomous navigation and digital twin construction in complex driving environments.

Related work, including radar-only odometry and multimodal sensor fusion, is discussed to contextualize the contributions of this research within the broader field of autonomous vehicle navigation [10], [12], [13], [14].

***Index Terms*—Radar Odometry, mmWave Radar, ICP, Doppler Velocity, Doppler Augmentation, Ego-Motion Estimation, Digital Twin**

I. Introduction

Accurate and reliable ego-motion estimation is a fundamental requirement for mobile robotic systems and autonomous vehicles solutions.

Traditionally or in most scenarios, this task has been accomplished using a combination of wheel encoders, inertial measurement units (IMUs), and GPS. In the most recent years, odometry has been implemented using vision (cameras) or LiDAR-based sensors, which offer dense information about the environment. However, these modalities often suffer from high cost and performance degradation under adverse conditions such as low illumination, fog, rain, or snow. As a result of this performance degradation, the accuracy and robustness of odometry are significantly limited and reduced, raising the demand for complementary sensing solutions which may offer a more robust solution in such scenarios.

Odometry provides the basis for localization, mapping, and navigation, serving as a core component in modern perception solutions. In most scenarios, odometry has been implemented using vision- or LiDAR-based sensors, which offer dense information about the environment. However, these modalities often suffer from performance degradation from high memory consumption and being under adverse conditions such as low illumination, fog, rain, or snow. In such scenarios, the accuracy and robustness of odometry are significantly limited and reduced, raising the demand for complementary sensing solutions. This demand is particularly critical at higher speeds, where LiDAR odometry may face challenges when the environment is not as optimal for this sensor usage, but radar velocity factors have been proven to enhance robustness and accuracy [14].

Millimeter-wave (mmWave) radar has emerged as a promising candidate to address these challenges. Due to its resilience to environmental variability, low cost and native capability to measure both range and velotic. Radar sensors are compact, cost-efficient, and resilient to weather and lighting variations, making them attractive for robotic and automotive applications. Unlike vision or LiDAR, radar directly measures range and Doppler velocity, providing unique information for motion estimation. Such Doppler-based measurements have already been demonstrated as effective for drect ego-motion estimation, showing that radar alone can support reliable vehicle odometry [12].

Nevertheless, radar data presents notable challenges: the resulting point clouds are sparse, noisy, and often subject to multipath reflections. Thus complicating the task of reliable odometry. These limitations hinder the direct application of a traditional scan-matching techniques, commonly used in LiDAR odometry. In which the assumption of a highly structured and dense point cloud is made. To overcome these challenges in unstructured environments, multimodal fusion strategies that include radar have been shown to significantly enhance robustness and accuracy [13], [14].

This work explores the use of using mmWave sensors

mounted in a vehicle with the goal of obtaining the ego-motion of the vehicle. To solve the challange presented by the sparsity and noise that is inate to data obtained from mmWave sensors, a combination of techniques is proposed. Presented in a pipeline that is easy to understand. The proposed workflow relies heavily in leveraging the Doppler effect and iterative closest point (ICP) alignment between submaps of frames to obtain a more accurate information for this purpose.

Instead of applying ICP globally, the key insight of this work is to track multiple clusters frame by frame, performing ICP iteratively on the aggregated submap and tracked clusters. By doing so, the sparsity of radar data is mitigated, and the robustness of the odometry estimation is improved. This approach enhances the stability and accuracy by focusing the motion on the "tragets" that are considered static in the environment and using them as a reference for the ego-motion estimation. This design choice aligns with recent findings showing that radar-driven odometry can achieve competitive performance when combined with Doppler velocity information [11]. Particularly when augmented with Doppler velocity information [14].

The full pipeline inclused a RANSAC-based Doppler filter to remove the dynamic point by making the assumption that the mayority of the detected points or reflections are static objects. The assumption that any reflection from a dynamic object o target will have a Doppler velocity closer to zero and different from the fitted curve model that RANSAC will provide.

The contributions of this work can be summarized as follows:

1) A radar ego-motion pipeline using mmWave sensors and an IMU for rotation, minimizing the hardware cost and system complexity.
2) Integration of Doppler velocity and RANSAC filtering improves the distinction between static and dynamic objects.
3) Submap aggregation to mitigate point cloud sparsity and improve alignment stability.
4) Object tracking via clusters to identify and filter dynamic objects from the ego-motion estimation.
5) Experimental validation using real-world data collected from a vehicle-mounted mmWave radar sensor.

II. Objective and Sub-Tasks

The main objective of this work is the development of a radar-based odometry system that estimates the vehicle ego-mmotion using mmWave radar sensors mounted at the fron of the vehicle setup in combination with an IMU for rotation compensation. That motivation behind this approach is to explore radar as a cost-effective and robust alternative to the existing solutions based in vision or LiDAR-based odometry, particularly in conditions where those have the tendency to fail. This builds on prior evidence that radar can support instantaneous ego-motion estimation through Doppler velocity cues [12].

The system processes radar point cloud data enriched with range, angle and the most important of all, Doppler velocity, aiming to extract accurate motion for ego-velocity or speed estimation. This to be able to recreate the trajectory of the vehicle and be able to use this information for SLAM applications.

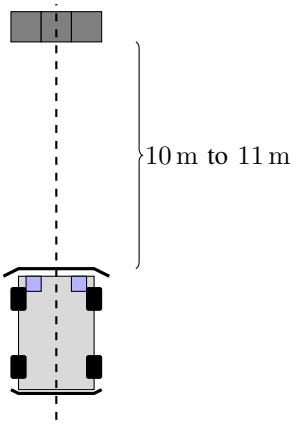


Fig. 1: Test scenario with dual front-mounted mmWave radar sensors.

The experimental setup is illustrated in Figure 1, where two mmWave radar sensors are mounted at the front of a vehicle, providing overlapping fields of view to enhance environmental perception. The dual-sensor configuration improves spatial coverage, reduces blind spots, and increases point density, resulting in more stable odometry processing compared to a single-sensor setup—an approach that aligns with multimodal methods for robust state estimation [13], [14].

By combining data from both sensors, including radial speed measurements derived from the Doppler effect, with the inertial measurement unit (IMU) for rotation compensation, the proposed system aims to ensure resilient odometry performance even under high-speed or degraded environmental conditions, where LiDAR-based odometry may fail [14].

Each radar perspective is processed independently and then merged into a single point cloud for further analysis. This integration creates a more robust and reliable input for the odometry estimation pipeline, enabling the evaluation of radar-based odometry performance in realistic driving scenarios.

A. Sub-Tasks

The research objective, together with the constraints of using a dual-radar sensor, implied several practical sub-tasks:

- Designing a modular pipeline to acquire and decode synchronized radar data from both sensors.
- Investigating suitable sensor configurations to balance field of view, chirp bandwidth, update rate, and detection density.
- Applying RANSAC filtering on Doppler velocities to reject dynamic points and outliers.
- Implementing clustering methods to structure radar detections and isolate relevant features.

- Optimizing the raw radar point cloud using additional information provided by the sensor itself (e.g., SNR, RCS, or range validity) to improve reliability before odometry processing.
- Integrating Doppler velocity information into the odometry estimation process.
- Employing submap aggregation to mitigate sparsity and improve stability.
- Performing ICP alignment between submaps aggregated from both sensors to mitigate sparsity and noise.
- Evaluating the influence of the dual-sensor arrangement on odometry accuracy and robustness.
- Validating the complete system on real-world driving scenarios.

As each sub-task builds upon the results of the previous one, the work followed an iterative and modular development approach, enabling gradual integration and continuous evaluation of the proposed system. The following sections detail the implementation and evaluation of each sub-task, culminating in a comprehensive radar-based odometry solution.

III. Chirp Sequence Configuration

To enable effective ego-motion estimation, the radar must be configured with a suitable chirp profile that balances range and velocity resolution. This is achieved through the configuration of Frequency-Modulated Continuous Wave (FMCW) chirps, which define how the radar sweeps frequencies over time to capture environmental information.

In millimeter-wave FMCW radar systems such as the IWR6843, a chirp refers to a signal whose frequency increases (or decreases) linearly over a defined bandwidth B during a fixed duration T_c. When this signal reflects off a target and returns to the radar, the delay between transmitted and received signals encodes range information. The Doppler shift between multiple received chirps in a frame encodes relative velocity. As seen in Figure 2, the reflected wave varies in phase and frequency depending on object motion.

Fig. 2: Radar 60-64GHz radio wave reflection.

Each chirp is defined by the following core parameters:

- Start frequency f_0
- Frequency slope S
- Chirp duration T_c
- Idle time between chirps T_{idle}
- Sampling rate f_s
- Bandwidth $B = S \cdot T_c$

From these parameters, the radar sensing capabilities are derived as follows:

- **Range resolution:**

$$\Delta R = \frac{c}{2B}$$

where c is the speed of light and B is the chirp bandwidth.

- **Maximum unambiguous range:**

$$R_{max} = \frac{cf_s}{2S}$$

where f_s is the ADC sampling rate and S is the frequency slope.

- **Velocity resolution:**

$$\Delta v = \frac{\lambda}{2N_c T_c}$$

where λ is the wavelength, N_c is the number of chirps per frame, and T_c is the chirp duration.

- **Maximum unambiguous velocity:**

$$v_{max} = \frac{\lambda}{4T_c}$$

assuming uniform chirp spacing.

These equations clearly show that both range and velocity resolution depend directly on the chirp design.

A. Chirp Design Strategies and Trade-offs

Radar applications differ in their emphasis on range vs velocity accuracy. Odometry requires both. We now examine key chirp configuration strategies and discuss their trade-offs.

(1) Full-Bandwidth Chirp (Single 4 GHz Sweep): A single chirp sweeping across the entire 4 GHz available bandwidth provides the finest possible range resolution. This is ideal for tasks requiring precise spatial localization of static objects.

Trade-offs:

- **Pros:** Excellent range resolution (ΔR minimized).
- **Cons:** High ADC sampling rates required. Limits maximum measurable range (R_{max}). Increases noise sensitivity. Not ideal for multi-radar systems due to potential mutual interference.

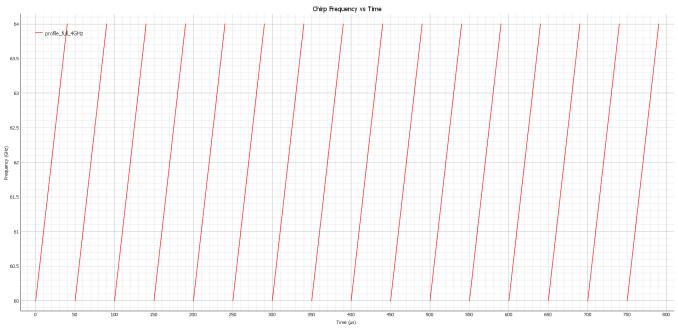


Fig. 3: Single 4 GHz chirp.

Conclusion: Best for high-resolution mapping in single-device systems. May suffer in multi-radar setups due to wide spectral overlap.

(2) Divided-Band Chirps (Multiple Shorter Sweeps): An alternative is to split the full 4 GHz bandwidth into multiple chirps with reduced individual sweep widths (e.g., 2 GHz). Each chirp provides moderate resolution.

Trade-offs:

- **Pros:** Reduced ADC load. Allows frequency diversity across chirps.
- **Cons:** Each individual chirp has worse ΔR than full sweep. However, averaging multiple bands improves robustness.

Conclusion: Balanced design. Useful in scenarios requiring robustness against channel noise or radar interference. Also reduces overlap when multiple radars are active.

(3) Mixed Chirp Strategy (Full-Band + Narrow-Band Chirps): Combining different chirp types in a single frame can provide both high range and velocity resolution. For example, use one full-band chirp for precise range, followed by multiple narrow-band chirps for velocity estimation.

Trade-offs:

- **Pros:** Leverages benefits of both strategies. Improves multi-parameter resolution without overwhelming hardware.

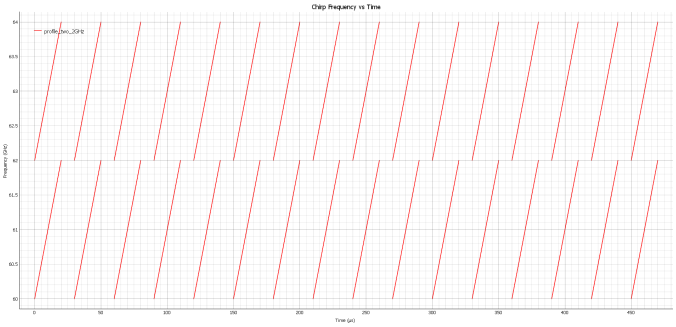


Fig. 4: Two 2 GHz chirps in the same frame.

- **Cons:** Increases configuration complexity. Requires advanced processing and careful interleaving.

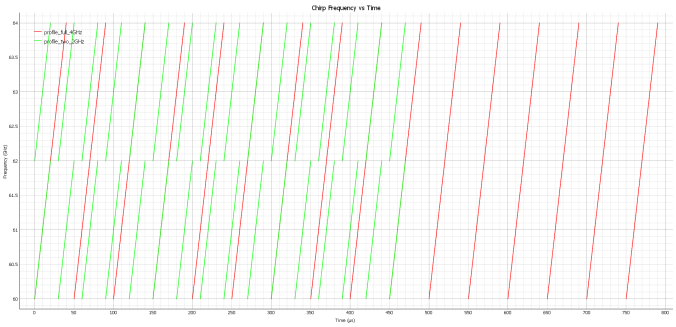


Fig. 5: Overlay of single 4 GHz chirp and two 2 GHz chirps.

Conclusion: Best suited for ego-motion and odometry applications. Can support multi-device environments with careful planning.

TABLE I: Comparison of Chirp Configuration Strategies

Chirp Strategy	Range Resolution	Velocity Resolution	Best Use Case
Full-Bandwidth Chirp (4 GHz)	High (Fine ΔR)	Moderate (Few chirps/frame)	Precise mapping in single-radar setups; spatial accuracy prioritized
Divided-Band Chirps (e.g., 2×2 GHz)	Moderate (per chirp), improved via averaging	Moderate to High (more chirps/frame possible)	Robust odometry in noisy or multi-radar environments
Mixed Chirp Strategy (Full + Narrow)	High (from full chirp), plus enhanced Doppler	High (from multiple narrow chirps)	Ego-motion estimation; combines benefits of both range and Doppler precision

The table above summarizes the key performance characteristics and use cases for each chirp configuration strategy.

The Full-Bandwidth Chirp maximizes spatial resolution, making it ideal for scenarios where high-fidelity mapping of nearby static objects is required—such as indoor SLAM or obstacle-rich environments. However, its wide spectral footprint and high ADC demand make it less suitable in systems where multiple radar devices operate simultaneously, as mutual interference becomes likely. The Divided-Band Chirp approach offers a middle ground. By allocating portions of the spectrum to separate chirps, it reduces system load and provides flexibility in processing. While individual chirps have lower range resolution, combining information across them improves detection stability. This strategy excels in distributed radar systems, such as in collaborative or multi-node sensor networks, where frequency coordination helps reduce ghost targets and interference. Finally, the Mixed Chirp Strategy

integrates the strengths of both previous approaches. A full-band chirp ensures accurate spatial anchoring, while narrow-band chirps provide temporal and Doppler refinement. This makes it the most adaptive option for applications like ego-motion estimation, where both precise localization and motion cues are critical. Its complexity, however, demands a more advanced signal processing pipeline and careful calibration to ensure synchronization between chirps.

In multi-radar environments, especially in close proximity (e.g., autonomous fleets or dual-radar vehicles), selecting chirp strategies that reduce spectral overlap is essential. Techniques like time-division multiplexing, slope diversity, and chirp staggering become critical tools to mitigate false detections and interference.

B. Chirp Configuration for Multi-Radar Systems

When deploying multiple radar devices (e.g., front-left, front-right), interference becomes a significant challenge. Simultaneous transmission using overlapping chirp frequencies can lead to ghost detections, false targets, and degraded signal-to-noise ratio. This due to the radio waves that are transmitter can cause collitions between each other creating this "ghost detections". There are some strategies that can be employed to avoid this phenomenom.

Recommended strategies:

- Use **non-overlapping frequency ranges** per device (e.g., 60–62 GHz on left, 62–64 GHz on right).
- Apply **interleaved chirps** in time: stagger chirps across sensors.
- Utilize **different slope values**: reduces likelihood of constructive interference.
- Match chirp **idle times** to ensure orthogonality.

These methods are essential to avoid ghost detections and preserve spatial coherence across sensors.

Conclusion: Chirp configuration in multi-radar environments requires trade-offs between resolution and isolation. Design must ensure orthogonality to avoid mutual interference.

The configuration of chirps in FMCW radar defines its core capabilities in range and velocity estimation. Depending on the target application — whether precision mapping, ego-motion tracking, or multi-device deployment — the choice of chirp bandwidth, duration, slope, and sequencing must be carefully made.

The following considerations are essential:

- Use wide-band chirps when range precision is critical.
- Use many chirps with long frame durations when velocity resolution is prioritized.
- Use divided-band or mixed strategies in multi-radar systems to balance robustness and prevent interference.

Careful design enables robust radar odometry, reliable environment mapping, and scalable multi-radar systems.

IV. IWR6843 Radar Interface and Configuration

The IWR6843AOPEVM development board from Texas Instruments features the IWR6843AOP, a high performance 4D mmWave FMCW radar sensor with Antenna On Package (AOP) design. Although IWR6843AOP is intended for industrial applications and its complementary chip, AWR6843AOP, for automotive applications, IWR6843AOP was used in this project because it is available in the form of this development board and the two chips are identical in terms of their functionalities, only differing in compliance with automotive industry [2]. Its small physical size, due to its AOP design, makes it an optimal choice for the desired mounting position, the go-kart's steering column.

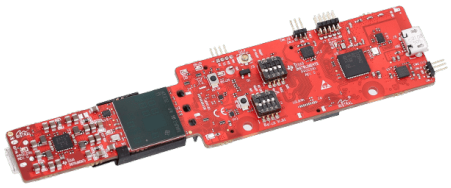


Fig. 6: IWR6843AOP sensor

The IWR6843AOP radar sensor operates within the frequency range of 60 GHz to 64 GHz and integrates 4 receive (RX) and 3 transmit (TX) antennas, radio frequency (RF) front-end stages, analog signal processing, and digital signal processing (DSP). It offers a wide range of communication interfaces including SPI, I2C, CAN-FD, UART and LVDS for raw data access and an Arm Cortex-R4F microcontroller for user-applications [3].

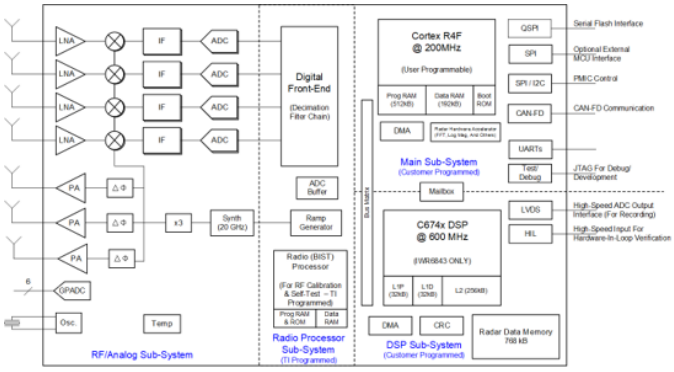


Fig. 7: IWR6843AOP internal Block Diagram

Texas Instruments offers various demo applications for the development board that utilize the internal microcontroller for showcasing the radar sensor's capabilities in different specialized scenarios. It was found out that most of them only use the radar sensor itself only for obtaining a point cloud, which points include spatial information in form of x, y, z coordinates and a radial speed information (the prior mentioned four dimensions of the sensor). Application-specific

processing of the point cloud itself is executed on an external computation device.

The main demo application ("mmWave SDK demo") allows a versatile customization of the radar sensor's operating parameters and its discrimination capabilities while outputting the point cloud via its UART interface, accessible via the on-board USB to UART converter. Although the demo seems to be intended to be used only for demonstration purposes, many projects based on Texas Instruments' mmWave radar sensors utilize it, because it poses a generic solution for obtaining (close to) real-time point cloud data from the sensor without prior development of a custom user-application for the radar sensor's internal microcontroller. This setup was therefore chosen for supplying the emergency braking system with data.

A. Utilizing the mmWave SDK demo

The "mmWave SDK demo" was developed by Texas Instruments for showcasing the abilities of their mmWave radar sensors. It consists of the radar sensor itself, supplying close to real-time point cloud data and an online tool for visualization of the raw output data and for the creation of a sequence of commands used for configuration of the radar sensor's operating parameters and output [4]. Due to the demo's simple structure and the radar sensor's generic output, the online application can be replaced by a custom application replicating the online tool's behavior for making use of the data.

In the demo application, the radar sensor opens two UART connections. One connection is bidirectional at a lower speed of $115\,200\,\mathrm{Baud}$ which is used to configure the radar sensor by sending the previously mentioned sequence of commands. The second connection is unidirectional, from the radar sensor to the receiver, at a higher speed of $921\,600\,\mathrm{Baud}$ and is used for outputting a constant data stream after the radar sensor received its configuration and a start command. As the data packets are encoded in a proprietary format and therefore need to be parsed prior to further processing, a custom software module was written in C++ and later ported to Python. The module handles the radar sensor's setup by sending a configuration file containing the sequence of initialization commands and the cyclic parsing of the encoded data packets. The sequence of commands was generated with Texas Instruments' online tool, as it provided graphical feedback while making the necessary compromises involved in setting up the radar sensor's operating parameters.

B. Sensor Data Output Format

As the data packets, containing the individual frames, are encoded in a proprietary format, they need to be parsed to allow for further processing. Each frame starts with a frame header and contains a number of TLVs (Type, Length, Value) in which the actual payload data is stored [4].

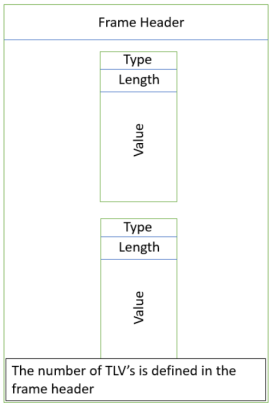


Fig. 8: Frame structure. From [6].

The frame's header has a total length of $40\,\mathrm{B}$ and starts with a fixed magic word that denotes the start of each frame. It also provides several other information in addition to the total packet length in bytes which is used to find the frame's end:

- Magic Word: This value indicates the start of a new header, meaning that it can be used as a starting point for processing each frame.
- Total Packet Length: Total number of bytes in the frame (including the header) which can be used to calculate the frame's end.
- Platform: Indicates the device type and can be used for validating the radar sensor type. In the case of the device used for this project (IWR6843AOP) the expected value is "$0xA6843$".
- Number of TLVs: Total number of TLV's that exist in that specific frame.

Value	Type	Bytes	Details
Magic Word	uint16_t	8	Output buffer magic word (sync word). It is initialized to {0x0102,0x0304,0x0506,0x0708}
Version	uint32_t	4	SDK Version represented as (MajorNum x 2^24 + MinorNum x 2^16 + BugfixNum x 2^8 + BuildNum)
Total Packet Length	uint32_t	4	Total packet length including frame header length in Bytes
Platform	uint32_t	4	Device type (ex 0xA6843 for IWR6843 devices)
Frame Number	uint32_t	4	Frame number (resets to 0 when device is power cycled or reset. Not when sensor stop/start is issued.)
Time [in CPU Cycles]	uint32_t	4	Time in CPU cycles when the message was created.
Num Detected Obj	uint32_t	4	Number of detected objects (points) for the frame
Num TLVs	uint32_t	4	Number of TLV items for the frame.
Subframe Number	uint32_t	4	0 if advanced subframe mode not enabled, otherwise the sub-frame number in the range 0 to (number of subframes - 1)

Fig. 9: Frame header format. From [6].

The frame contains one or more TLVs after its header. Each TLV has a header itself in which it specifies its length and which type of data (point cloud, doppler heatmaps, statistics, ...) is contained inside. Each TLV type needs to be decoded differently, as it represents a different type of data.

Value	Type	Bytes	Details
Type	uint32_t	4	Indicates types of message contained in payload.
Length	uint32_t	4	Length of the payload in Bytes (does not include length of the TLV header)

Fig. 10: TLV header format. From [6].

C. Sensor-Tuning

As some operating parameters influence each other, their selection must be done carefully while observing the influence of the trade-offs involved. This could be referred to as ”sensor tuning” and is a critical step because it directly impacts the system's accuracy and performance.

The following operating parameters can be tuned:

- Frame rate
- Range resolution
- Maximum unambigious range
- Maximum radial velocity
- Radial velocity resolution

Tuning these operating parameters introduces trade-offs by influencing each other in the following ways: The resulting

Tuning Parameter	Effect on Performance	Related HW Block	Trade-Off
Frame Rate	Higher FPS = faster updates but more processing load	C674x DSP, Radar Data Memory	Higher FPS reduces maximum range
Range Resolution	Higher resolution = better object separation	ADC, 1D FFT (Range FFT)	Higher resolution reduces max range
Maximum Range	Determines farthest detectable object	RF Front-End, PA, LNA, ADC	Higher range lowers resolution
Radial Velocity Resolution	Improves speed accuracy	DSP, 2D FFT (Doppler FFT)	Higher resolution requires more chirps
Maximum Radial Velocity	Detects fast-moving objects	Chirp rate, TX Antennas, 2D FFT	Higher max velocity reduces resolution

TABLE II: Radar System Tuning Parameters and Trade-offs

overall accuracy of the velocity and distance measurements is again dependent on these operating parameters:

- **Radial velocity accuracy:** A fine balance between velocity resolution and frame rate must be maintained to ensure precise Doppler shift measurements. Lower resolution results in rounded velocity values, while an excessively high frame rate may introduce computational bottlenecks.
- **Distance accuracy:** Optimizing range resolution and maximum range ensures that detected objects are positioned accurately within the environment. Increasing range often sacrifices resolution, leading to potential inaccuracies in close-range detections.
- **Signal processing considerations:** The FFT calculation parameters directly affect both range and Doppler calculations, influencing the ability to distinguish between objects and detect small velocity variations.

This shows that finding exact values for the operation parameters by adjusting them while carefully watching their influences is crucial and heavily dependent on the particular application. The test scenario required a frame rate of approximately 30Hz to balance responsiveness and computational load, together with sufficient range and velocity coverage to capture typical vehicle dynamics.

The resulting configuration yielded the following operating parameters:

- Frame rate: $30\,\mathrm{f\,s^{-1}}$
- Range resolution: $0.047\,\mathrm{m}$
- Maximum unambiguous range: $9.78\,\mathrm{m}$
- Maximum radial velocity: $8.01\,\mathrm{m\,s^{-1}}$
- Radial velocity resolution: $0.51\,\mathrm{m\,s^{-1}}$

Tuning and choosing the sensor's parameters carefully is extremely important as it defines the accuracy of therefore influences the reliability of the entire radar system. Fine-tuning these settings ensures that the sensor operates optimally, enabling more precise self-speed estimation and overall system performance. The accuracy of radial speed estimation and distance measurements depends directly on the tuning of these parameters. A poorly configured sensor can result in erroneous velocity estimations, unreliable object detection, or excessive noise in Doppler measurements. An example of the influence of the selection of the correct parameters on the output point cloud of the radar sensor can be found in Fig. 12.

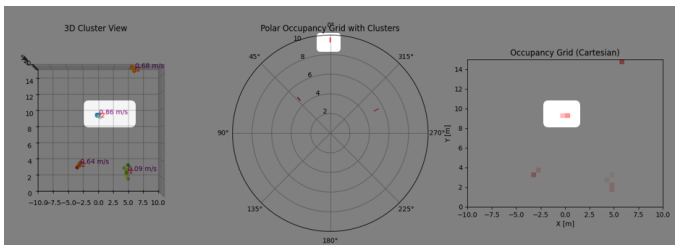


Fig. 11: Output of the IWR6843AOP prior sensor tuning.

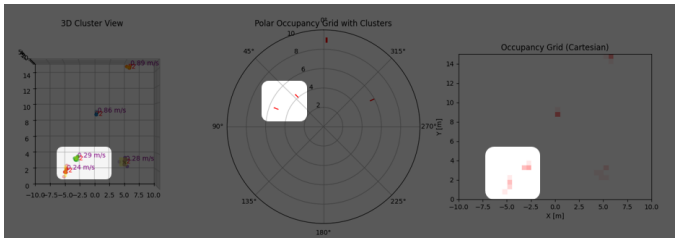


Fig. 12: Output of the IWR6843AOP after sensor tuning.

D. Dual Sensor-Tuning

In addition to the individual calibration of each radar, further adjustments were required to enable a dual-sensor configuration with overlapping fields of view. The sensors were physically rotated around the Z-axis to widen the horizontal coverage and tilted by $15°$ around the X-axis to prevent ground reflections and reduce clutter in the point clouds.

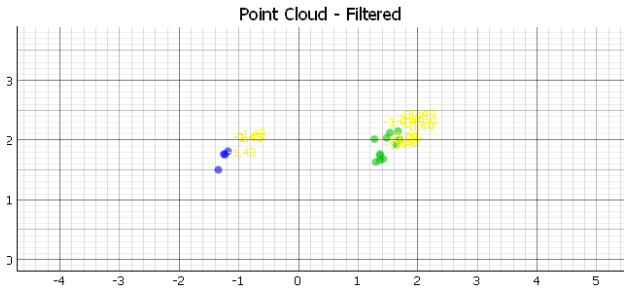
1) Chirp Calibration with Frequency Shift: Although both sensors operated under the same high-level parameters, their chirps had to be tuned to avoid mutual interference and ghost detections. To achieve this, each radar was configured to occupy only 2 GHz of the available 4 GHz bandwidth, introducing a frequency shift between the sensors. The resulting chirp design was derived using the Texas Instruments *mmWave Sensing Estimator* tool [5], which provides interactive validation of scene-dependent parameters, chirp design, and power consumption estimates. This configuration allowed the sensors to operate simultaneously without cross-talk while maintaining sufficient range resolution and velocity accuracy.

2) Geometric Transformations: To merge both radar outputs into a consistent reference frame, rigid-body transformations were applied to compensate for rotation, tilt, and translation offsets between the sensors. Each radar detections were first processed in its local coordinate frame and then transformed into the vehicle-centric frame before fusion into a single point cloud.

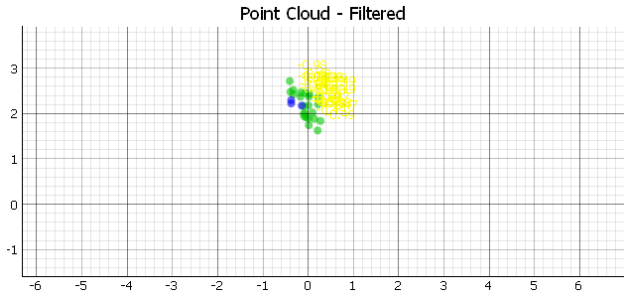
Figures 13–15 illustrate the effect of this calibration process at different processing stages:

- **Raw point clouds:** Before calibration (Fig. 13a), the detections from each sensor appear misaligned, producing duplicated landmarks. After applying the transformations (Fig. 13b), both perspectives are fused into a coherent scene.
- **Clustered view:** When clustering is applied, the uncalibrated data (Fig. 14a) shows inconsistent cluster centers, while the calibrated version (Fig. 14b) produces compact and aligned clusters.
- **RANSAC Doppler fitting:** Similarly, Doppler–azimuth consistency improves after calibration. Uncalibrated detections (Fig. 15a) yield noisier distributions, while the calibrated outputs (Fig. 15b) produce smoother fits with fewer outliers.

The transformed data thus provides a coherent and stable input for the odometry estimation pipeline, ensuring that static landmarks are consistently aligned across both sensors.

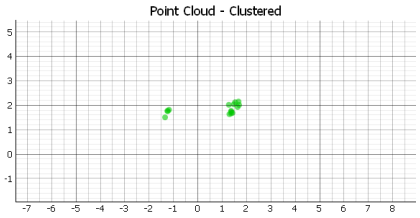


(a) Raw point cloud before calibration

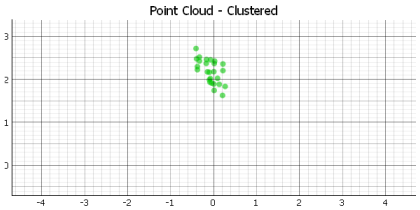


(b) Raw point cloud after calibration

Fig. 13: Dual-sensor raw detections before and after applying geometric transformations.

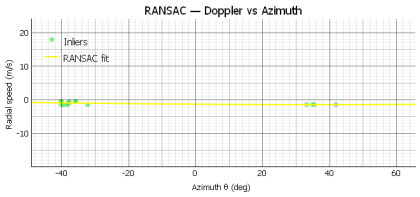


(a) Clustered detections before calibration

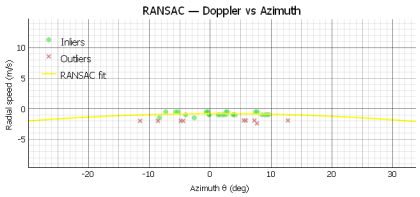


(b) Clustered detections after calibration

Fig. 14: Effect of calibration on cluster alignment across sensors.



(a) RANSAC Doppler fitting before calibration



(b) RANSAC Doppler fitting after calibration

Fig. 15: Improvement in Doppler-azimuth consistency after calibration.

V. Pipeline Implementation: Modules Implemented and Mathematical Explanation

To reliably interpret radar sensor data for static object detection and ego-motion estimation, a modular processing pipeline was developed. This pipeline supports both single-radar and dual-radar configurations, in both cases the inclusion of an Inertial Measurement Unit (IMU) is needed to compensate for rotational motion of the platform. The modular design allows adaptation to different vehicle setups and deployment conditions.

Sensor Data Preprocessing

In the preprocessing of the radar data, there are two scenarios that were implemented for this project, the single-radar setup and the dual-radar setup with IMU integration. In both

radar setups, radar data is received over UART in the form of raw frames containing point cloud information. Each point includes its spatial coordinates (x, y, z), radial velocity v_r, side-point info; this information comes with signal-to-noise ratio (SNR) and noise ration (dB) values that indicate the quality of each detection and range profile data.

The raw bytes are decoded [5] and structured into frame objects, which are then passed through a Physical Transformation module. This step converts from the sensor frame coordinates into the vehicle local frame, accounting for radar mounting position and orientation. This can be consider a rigid body transformation involving rotation and translation matrices. However is really important that this process of rigid transformation is done, as the data that we obtain from the sensor does not know where it is in the vehicle, creating a missalignment between the radar data and the vehicle frame of reference.

Dual-Radar Configuration: In the dual-radar configuration, two front-facing sensors are mounted symmetrically on the left and right sides of the vehicle, providing overlapping fields of view. This arrangement expands the effective coverage and reduces blind spots, offering a broader perception range compared to a single-radar setup.

The use of two radars requires careful calibration to avoid interference effects, such as ghost detections or false targets, and to ensure consistent alignment of their outputs. When properly tuned, the dual-radar configuration produces a denser and more reliable point cloud, serving as a stronger foundation for subsequent stages of the odometry pipeline.

Frame Aggregation

Radar point clouds are naturally sparse, especially at longer ranges. To improve stability and robustness, consecutive frames are aggregated into a local submap. This approach increases point density, reduces the effect of random noise, and provides a richer structure for downstream tasks such as clustering and alignment. The aggregation is performed after IMU-based alignment, ensuring that the fused frames are geometrically consistent.

Filtering and Self-Speed Estimation

Once points are aggregated and aligned, the pipeline performs filtering in two stages:

- **Coordinate and SNR Filtering:** Removes points outside valid spatial boundaries or below a configurable SNR threshold.
- **Velocity-based Filtering:** Applies a Doppler threshold to remove points with radial velocities below or above predefined limits. This eliminates static clutter and extreme outliers, leaving only detections within the valid motion range.

From the remaining points, the vehicle self-speed is estimated using the Doppler measurements. A Kalman filter is then applied to smooth fluctuations and provide a stable velocity estimate, which is reused in subsequent stages of the odometry pipeline.

Clustering and Outlier Rejection

The filtered detections are organized into clusters to provide structure to the radar scene. A two-stage clustering strategy is employed to gradually refine the set of candidate targets:

- **Stage 1 – Permissive Clustering:** A loose grouping is applied using the DBSCAN algorithm, which is well-suited for radar point clouds since it does not require the number of clusters to be specified in advance and can naturally separate sparse detections. At this stage, the parameters are set permissively, allowing loosely associated points to be grouped into preliminary clusters so that potential targets are not discarded prematurely.
- **Stage 2 – Strict Clustering:** The preliminary clusters are reprocessed with stricter DBSCAN parameters, enforcing tighter geometric proximity between points. This refinement step eliminates loosely connected points and isolates only the detections that are spatially consistent, resulting in compact and reliable target clusters.

This hierarchical approach ensures that targets are first captured broadly and then refined to retain only geometrically close detections, which are more suitable for downstream odometry estimation. Additionally, RANSAC-based outlier rejection is performed prior to clustering, further reducing the influence of inconsistent or spurious points on the cluster formation.

DBSCAN's ability to handle varying cluster shapes, adapt to density variations, and explicitly label noise points makes it particularly advantageous in radar applications, where detections are inherently sparse, noisy, and often non-uniformly distributed.

ICP Alignment and Ego-Motion Estimation

The final stage of the pipeline estimates vehicle motion by applying the Iterative Closest Point (ICP) algorithm for frame-to-frame alignment. ICP iteratively minimizes the distance between corresponding points across consecutive frames, computing the rigid-body transformation that best aligns them.

In this project, ICP was applied under two distinct configurations for analysis:

- **Global ICP:** The entire aggregated point cloud from consecutive frames is used for alignment. This approach provides a direct estimate of ego-motion but is sensitive to outliers and dynamic objects present in the scene.
- **Cluster-based ICP:** Instead of aligning full point clouds, ICP is applied selectively on clusters identified as static targets. By focusing on geometrically consistent regions, this method looks to improve robustness against moving objects and to enhance the stability of the motion estimate.

The outcome of ICP is a rigid transformation that represents the vehicle's ego-motion between frames, serving as the foundation for radar-based odometry.

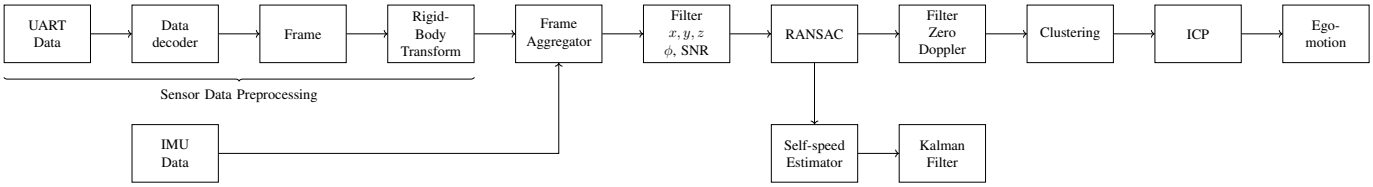


Fig. 16: Single Radar Pipeline

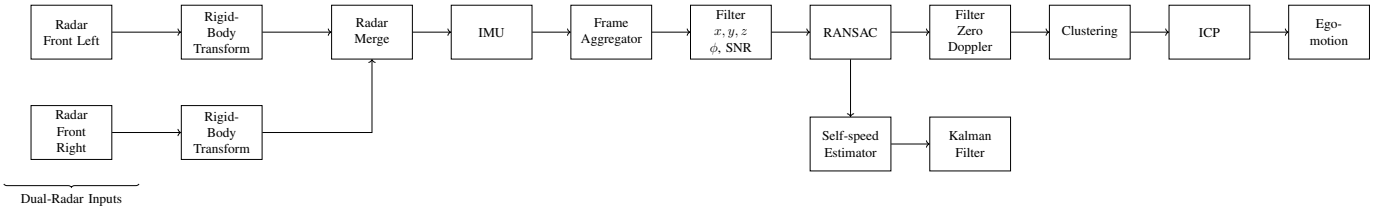


Fig. 17: Dual Radar Pipeline

A. Sensor Data Preprocessing

Each radar sensor in the dual-radar configuration is physically mounted with a specific orientation and tilt relative to the vehicle's forward direction. To ensure a common frame of reference for all points, it is necessary to compensate for:

- **Yaw rotation**: due to angled placement (±30°) of the radar modules.
- **Pitch tilt**: due to upward mounting tilt (15°), which must be compensated to recover horizontal geometry.
- **Sensor offset**: due to the physical separation of the radar sensors in the horizontal axis (X-axis).

A yaw rotation R_{yaw} was first implemented, followed by a pitch correction R_{pitch}, and finally a translation vector $\vec{t}$ was applied to account for the physical position of the sensors with respect to the vehicle's coordinate origin.

The full transformation can be expressed as:

$$T_{\text{veh}} = R_{\text{yaw}} \cdot R_{\text{pitch}} \cdot \vec{p}_{\text{radar}} + \vec{T} \tag{1}$$

Where:

- $\vec{p}_{\text{radar}}$ is a radar point in sensor coordinates.
- R_{yaw} is a 2D rotation around the vertical axis.
- R_{pitch} corrects for the upward sensor tilt.
- $\vec{T}$ is the translation vector.

Let (x, y, z) be the original point from the radar frame, with x to the right, y forward, and z upward. The transformations applied are as follows:

a) Yaw Correction (Z-axis Rotation): The radar sensors are rotated relative to the vehicle frame:

- Radar A (Left): Mounted at $+30°$ yaw $\Rightarrow$ compensated with $-30°$ rotation.
- Radar B (Right): Mounted at $-30°$ yaw $\Rightarrow$ compensated with $+30°$ rotation.

The 2D rotation in the XY-plane is defined as:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where $\theta = \pm 30°$ depending on the sensor.

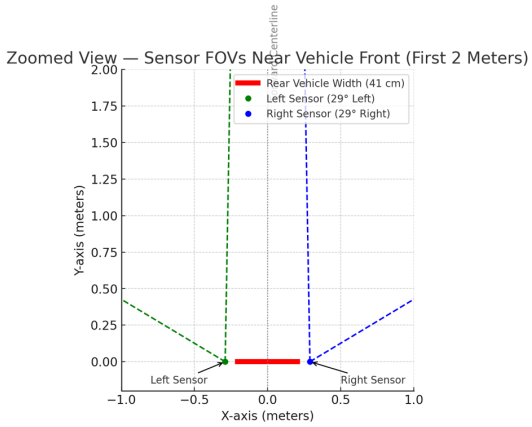


Fig. 18: Yaw rotation of sensors around the Z-axis (±30°).

b) Pitch Compensation (X-axis Rotation): Since the radar sensors are tilted upward by 15°, a corrective rotation around the X-axis is applied to bring the points back to a horizontal perspective:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where $\phi = -15°$ (negative to reverse the upward tilt).

c) X-Axis Offset Compensation: After rotation, each sensor's point cloud was translated along the X-axis to align with the vehicle's center:

- Radar A: $x \leftarrow x - 0.32$ meters
- Radar B: $x \leftarrow x - 0.28$ meters

This translation ensured both sensors were aligned in a common vehicle-centric frame.

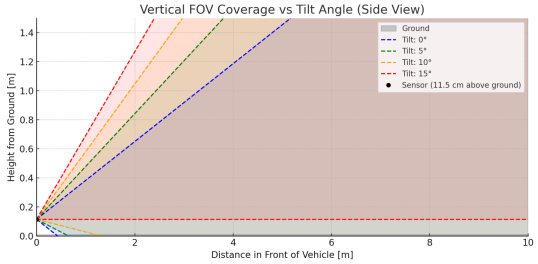


Fig. 19: Pitch compensation for 15° upward tilt.

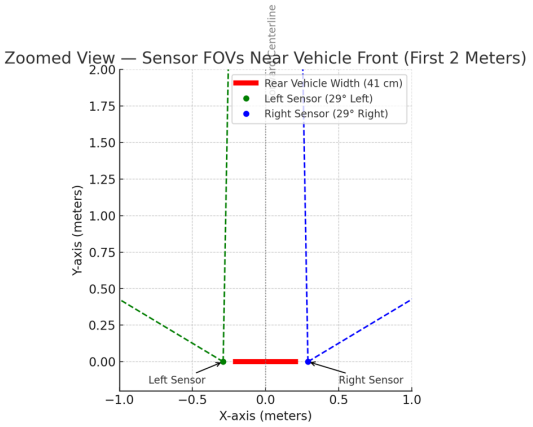


Fig. 20: Final extrinsic configuration combining yaw, pitch, and translation.

d) Impact of Transformations: The importance of these corrections becomes evident when comparing raw and transformed data when the system data is unified in a single data set. Without transformations, detections from the left and right radars appear misaligned, producing duplicated or inconsistent clusters. After applying yaw, pitch, and translation compensation, the fused point cloud exhibits coherent structures, with static landmarks consistently overlapping across sensors.

Figures 21–26 illustrate this process, showing raw point clouds, clustered representations, and RANSAC fits before and after calibration. This information is of a person standing at 2 meters away form the vehicle, the process was decided as it was easy to visualize and to validate what was detected. The transformed data serves as a stable basis for subsequent steps in the odometry pipeline, such as ICP-based submap alignment.

e) Summary of Extrinsics:

- **Left radar:** yaw $+30°$, pitch $-15°$, translation $(-0.32, 0, 0)$.
- **Right radar:** yaw $-30°$, pitch $-15°$, translation $(-0.28, 0, 0)$.

This extrinsic calibration ensures that dual-radar data is expressed in a coherent vehicle-centric frame, which is critical for downstream modules such as clustering, odometry, and obstacle tracking.

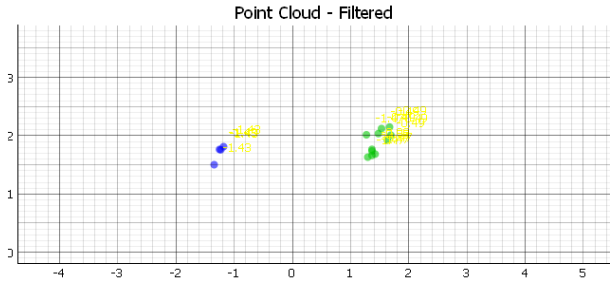


Fig. 21: Raw dual-sensor point cloud before geometric transformations.

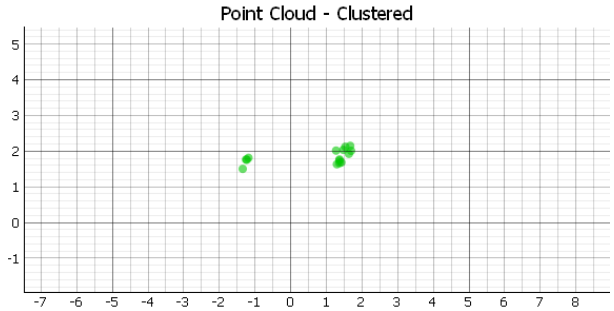


Fig. 22: Clustered detections from both radars before transformation.

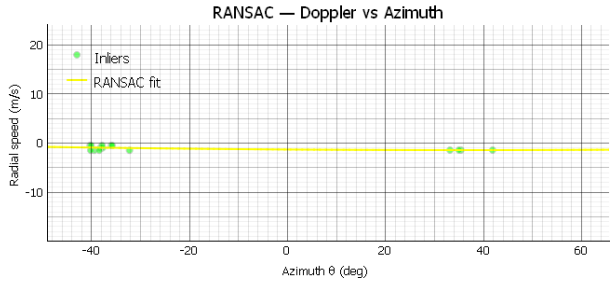


Fig. 23: RANSAC fit on Doppler velocities before transformation.

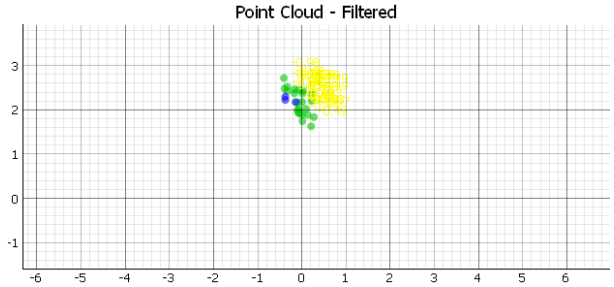


Fig. 24: Fused point cloud after geometric transformations.

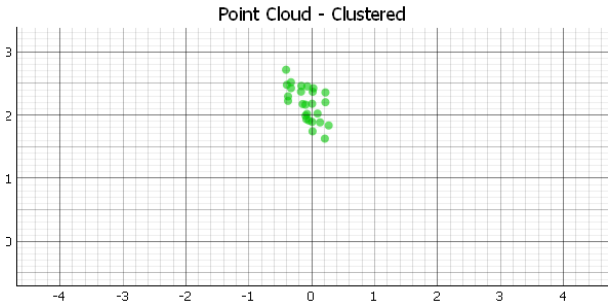


Fig. 25: Clustered detections after transformation, showing improved alignment.

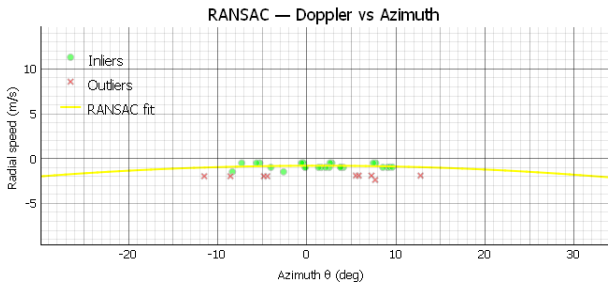


Fig. 26: RANSAC fit on Doppler velocities after transformation.

B. Frame Aggregator

The decoded frames are passed into a frame aggregator stage as a first step, which utilizes data aggregation to reduce possible data sparsity of individual frames. Data aggregation involves combining multiple consecutive data sets to create a more comprehensive and reliable dataset for processing. This approach increases both useful information and noise, but ultimately enhances the quality of radar point clouds for object detection and tracking stability. For point clouds from radar sensors, aggregating multiple frames enhances data consistency and density, improving the system's ability to reconstruct objects and detect motion patterns.

1) Single Frame: Using radar point cloud data from a single frame has inherent limitations, which can lead to incomplete or misleading detections:

- A single frame may not capture enough points, leading to failed detections or incomplete object reconstruction.
- Limited point data can cause objects to appear fragmented or indistinguishable from noise.

Consequently, it is difficult to precisely rebuild or map complex objects due to the limited points in the point cloud and information contained in a single radar frame, which results in ambiguity and uneven detection performance. This ambiguity can result in the erroneous detection of a large object or the merging of two different objects into one, as the clustering or detection algorithm may malfunction, not due to inherent flaws in the algorithm itself, but rather due to the inherent flaws in the data. This phenomenon occurs when two or more objects are in close proximity from each other, but

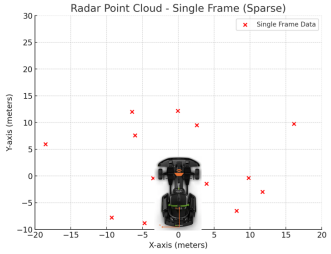


Fig. 27: Single Frame visualization.

can not be distinguished from each other because of the lack of points in a single frame.

To mitigate these issues, frame aggregation can be leveraged, an approach supported by the Law of Large Numbers (LLN). By accumulating multiple frames, the impact of random variations can be reduced, such as:

$$\frac{\sigma^2}{N} \tag{2}$$

This leads to a more statistically stable representation of objects; which means that the larger the sample is, the less effect the noise will have. Additionally, variance reduction helps to filter out erroneous detections while improving the reliability of motion estimation.

2) Multiple Frames: In contrast, the aggregation of multiple frames can significantly enhance object detection capabilities. Using this technique the algorithm can be tricked, by making it believe that there is more information that it was actually obtained from the single frame. This comes with the drawback that there will be some older data inside the processed frame.

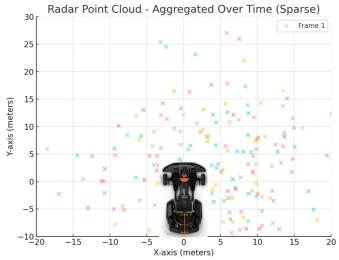


Fig. 28: Frame Aggregation visualization.

This aggregation approach improves the accuracy of velocity and trajectory estimations, strengthens object identification's resilience against transient noise, and reduces problems related to sparse data. It was implemented by using a stack with a fixed size, executing a "pop" operation prior to the insertion of the new frame at the stack's end. The point cloud containing the points of all frames stored in the frame aggregator is then passed to the next stage.

C. x,y,z,SNR Filter

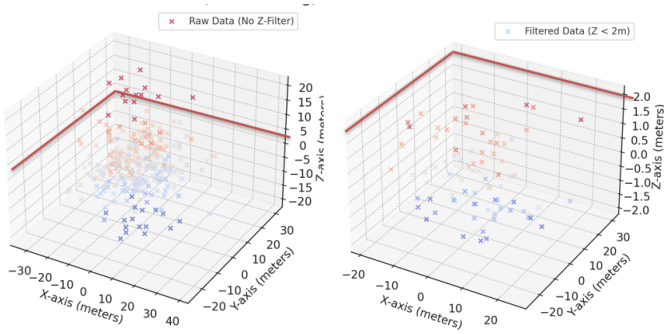
The point cloud's points are passed through a first static filtering stage to remove points caused by noise, clutter or

targets outside of the area of interest before being used for further processing. This static filtering stage consists of four different filters, filtering out points by different attributes:

1) Filtering by SNR: All points with a SNR lower than $12\,\mathrm{dB}$ are filtered out to remove points with a low signal and those that might be caused by noise or clutter.
2) Filtering by z coordinate: All points below a z value of $0\,\mathrm{m}$ and above $2\,\mathrm{m}$ are filtered out to remove points caused by the ground or the ceiling.
3) Filtering by y coordinate: All points below a y value of $0.3\,\mathrm{m}$ are filtered out to remove points created by the driver's feet.
4) Filtering by ϕ: All points with an azimuth bigger than $85°$ are filtered out to remove points that are outside the area of interest.

Note: The origin of the points' coordinate system is the sensor itself, so a coordinate of $(0\,\mathrm{m}, 0\,\mathrm{m}, 0\,\mathrm{m})$ *is essentially at the sensor's mounting position and therefore approx.* $0.3\,\mathrm{m}$ *above the ground.*

As the static filtering stage only keeps points which are relevant in terms of there spatial position and SNR for the following stages, it effectively decreases the computation time of each frame and prevents the following stages from processing invalid data. The filtered point cloud is then passed to the next stages, the self-speed estimator and the dynamic filtering stage.



(a) Input (b) Output

Fig. 29: Example visualization of the input and output when filtering by the value of the z coordinate.

Static vs Non-static objects: In the detected data there will always be static or moving objects in the field of view, this is more prominent when the sensor is mounted in a moving frame. In this case every thing is moving from the perspective of the sensor, so how does the differentiation happens to identify statis vs non-static objects, this is simple, by leveraging on the doppler effect concept. Using the doppler effect and understanding it is crutial for this step, as it is what will allow the algorithm to focuse the processing only in the static targets that are detected.

In this case, making the assumption that the sensor is mounted in a vehicle that goes 2 meters per second. With this in consideration, the assumption that all static targets should have a radial speed similar to the 2 meters per second. This speed cannot be the same as the angle of detection can affect, the accuracy that may be obtained through the chirp configuration, etc. Multiple variables can be added into this "equation", but the assumption that the static targets measured radial speed is close to the vehicle speed we can know which target to keep and which target to discard with only this paramter.

However that leaves us with the problem of how to group such targets. At this point the RANSAC algorithm becomes useful, as it will do the grouping for us, by ussing the radial speed.

D. RANSAC-Based Motion Filtering

A critical step in radar-based odometry is the separation of static landmarks from dynamic objects. This distinction ensures that ego-motion is estimated using only consistent reference points. To achieve this, a RANSAC-based filtering method is applied to the relationship between azimuth angle θ and Doppler velocity v_d.

Principle: When the radar is mounted on a moving platform, static objects do not appear motionless. Instead, they exhibit Doppler velocities that depend on the vehicle's own speed and the azimuth angle of detection. This results in a predictable curve when Doppler velocity is plotted against azimuth. Dynamic objects (e.g., moving vehicles or pedestrians) deviate from this pattern.

RANSAC (Random Sample Consensus) is used to robustly fit a polynomial model to the Doppler–azimuth distribution of static points. Inliers to the model are considered static reflections, while outliers are classified as dynamic.

Mathematical Model: The relationship between azimuth angle and Doppler velocity is modeled as a quadratic function:

$$v_d = a\theta^2 + b\theta + c.$$

Let $\mathbf{X}$ be the azimuth vector and $\mathbf{y}$ the Doppler velocities:

$$\mathbf{X} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} v_{d1} \\ v_{d2} \\ \vdots \\ v_{dn} \end{bmatrix}.$$

RANSAC estimates the coefficients $\mathbf{w} = [a, b, c]^T$ by iteratively:

- Sampling minimal subsets of (θ, v_d) pairs,
- Fitting the polynomial model,
- Counting inliers with residual error below a threshold,
- Selecting the model with the largest consensus set.

This approach is robust to noisy points and moving targets that would otherwise distort the fit.

Integration into the Pipeline: The RANSAC-based motion filter is applied after coordinate and Doppler thresholding. It outputs:

- **Inliers** - points likely to be static, used for clustering and ICP alignment,
- **Outliers** - points likely dynamic, excluded from ego-motion estimation.

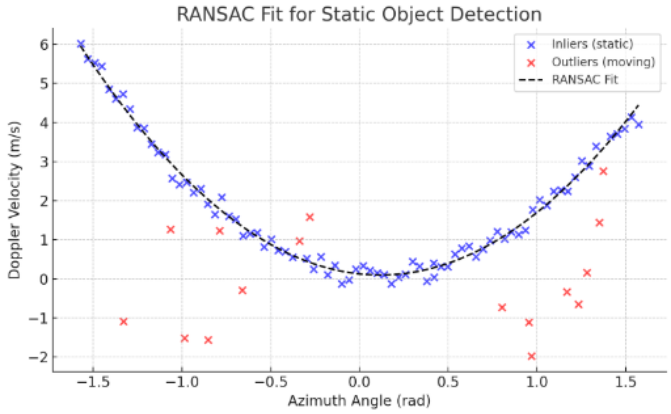


Fig. 30: RANSAC fit of Doppler vs. azimuth. Inliers = static (blue), Outliers = moving (red).
Note: This figure uses simulated data to illustrate the concept.

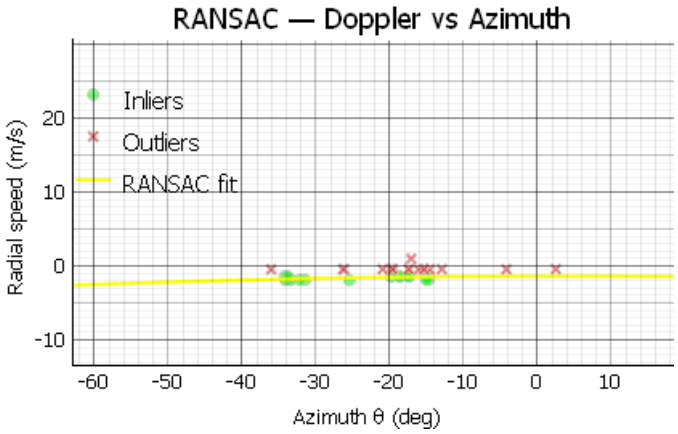


Fig. 31: RANSAC fit of Doppler vs. azimuth. Inliers = static (green), Outliers = moving (red).
Note: This figure shows real data when the vehicle is moving and a dynamic target is present.

Role of Doppler in Static vs. Dynamic Separation: For a vehicle moving at velocity v_e, static reflections follow a Doppler distribution determined by the projection of v_e along the radar line of sight. For example:

- A static object directly in front exhibits $v_d \approx v_e$,
- A static object at 90° azimuth exhibits $v_d \approx 0$,
- Intermediate azimuths follow a smooth transition between these extremes.

Dynamic objects violate this pattern, appearing as outliers in the Doppler-azimuth plane. This behavior is illustrated first on simulated data in Fig. 30, and then validated on real measurements in Fig. 31, where the presence of a moving target is evident. The corresponding clustered point cloud is shown in Fig. 32, where each object is assigned a unique identifier and centroid. Finally, the physical scene is provided in Fig. 33, which contextualizes the detected target in the real environment.

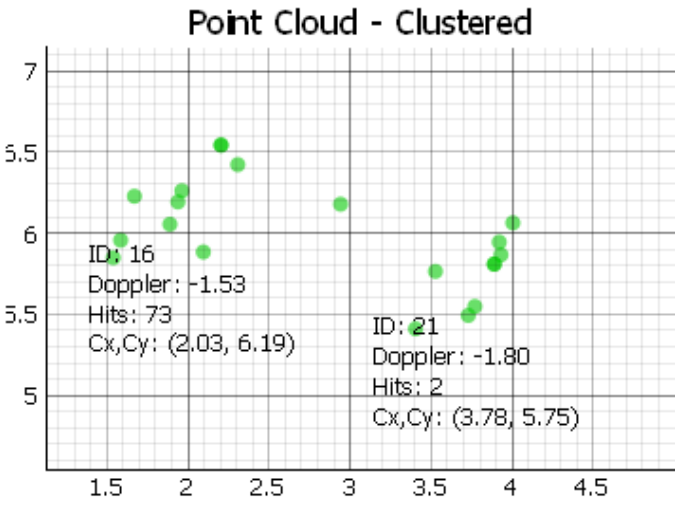


Fig. 32: Clustered point cloud corresponding to the RANSAC analysis in Fig. 31. Each cluster is annotated with ID, Doppler average, and centroid coordinates.

Fig. 33: Video frame at $t = 15$ seconds, showing the target in the scene. This frame corresponds to the radar data used in Figs. 31 and 32.

Benefits of RANSAC-Based Filtering:

- Robust against noise and measurement errors,
- Separates dynamic actors from static landmarks without prior labeling,
- Provides stable input for clustering and prevents moving objects from corrupting ego-motion estimation.

The output of this module is a refined point cloud in which only consistent static reflections contribute to downstream clustering and ICP alignment. In this system, RANSAC therefore plays a dual role: it filters out dynamic objects while simultaneously delivering a coherent static inlier set that underpins clustering stability and ICP-based odometry. As errors in static-dynamic separation would otherwise propagate downstream, this step is foundational for trajectory accuracy.

The integration of RANSAC-based motion filtering further ensures that odometry remains resilient in challenging scenarios, such as cluttered urban environments or mixed traffic conditions, where dynamic objects may dominate the scene. By anchoring estimation on reliable static structures, the system achieves greater robustness and consistency—critical factors for autonomous navigation. Additionally, maintaining a coherent model of the static Doppler pattern allows for a more accurate estimation of the vehicle's own self-speed, reinforcing the central role of RANSAC in radar-only odometry.

E. Self-speed Estimator

The vehicle's self-speed plays a crucial role in distinguishing between stationary and moving objects, as it enables the removal of dynamic points in later stages of the pipeline. It therefore needs to be determined in a reliable way. Traditional approaches usually utilize external sensors such as wheel encoders, Global Positioning System (GPS), or Inertia Measurement Units (IMUs). In this project, a technique was developed that determines the vehicle's self-speed only by processing the point cloud's data. An angle ϕ_p was defined between each individual point of the point cloud and the radar sensor's centerline:

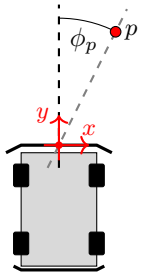


Fig. 34: Definition of the angle ϕ_p

Scalar Ego-Speed Estimation: The first stage estimates the forward speed of the vehicle under the assumption of motion along the $+y$ axis. For each radar detection, the angle ϕ_p relative to the sensor's forward axis is defined (see Figure 34):

$$\phi_p = \arctan\left(\frac{x_p}{y_p}\right).$$

The observed radial velocity of a static point p depends on the vehicle speed v_0 and follows a cosine law:

$$v_{r,p}(v_0, \phi_p) = -v_0 \cdot \cos(\phi_p).$$

Since no point lies exactly on $\phi = 0°$ and noise is present, all available inlier points (after RANSAC filtering) are used to fit a regression curve $v(\phi)$ to the measured pairs $(\phi_p, v_{r,p})$. The fitted value at $\phi = 0°$ provides the best estimate of the scalar vehicle speed. A least-squares polynomial regression of order 2 is sufficient, since $\cos(\phi)$ can be locally approximated in the domain $\phi_p \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ by:

$$\cos(\phi_p) \approx a\phi_p^2 + b.$$

Generalized 2D Ego-Velocity Estimation: The scalar approach assumes purely forward motion. To extend the model, the Doppler equation for a static point observed at angle θ is introduced:

$$v_r(\theta) = v_x \cos(\theta) + v_y \sin(\theta),$$

where v_x and v_y are the ego-velocity components along the x and y axes of the vehicle frame. This equation is linear in v_x and v_y, which allows solving for both components simultaneously using a least-squares approach across multiple points:

$$\begin{bmatrix} v_{r,1} \\ v_{r,2} \\ \vdots \\ v_{r,n} \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & \sin(\theta_1) \\ \cos(\theta_2) & \sin(\theta_2) \\ \vdots & \vdots \\ \cos(\theta_n) & \sin(\theta_n) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}.$$

This formulation naturally integrates into the existing pipeline: RANSAC provides inlier points representing static structures, and the least-squares solution yields robust estimates of both forward and lateral velocity components.

Practical Considerations: Test runs demonstrated that the radar-only ego-velocity estimate closely followed the reference vehicle speed with a small static offset. However, fluctuations occurred in cases with sparse point clouds. This effect was mitigated by tuning the frame aggregation window and applying a Kalman filter after the regression stage, providing a stable and continuous self-speed estimate for downstream filtering and odometry estimation.

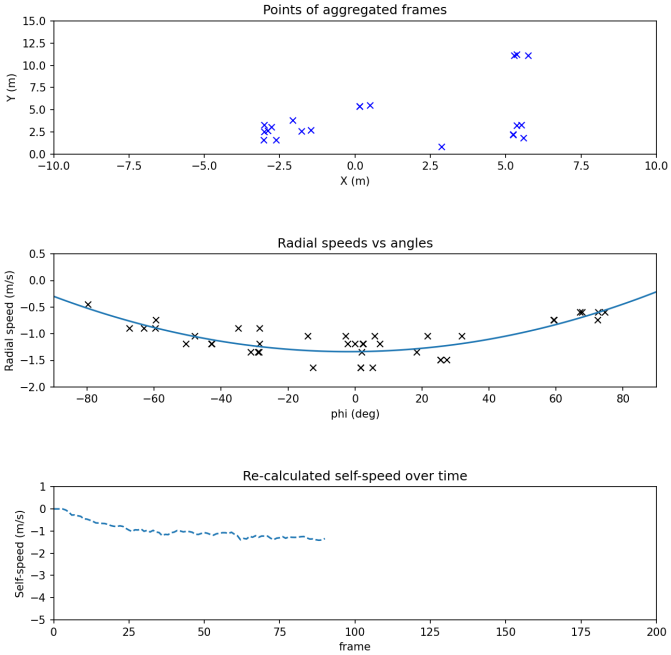


Fig. 35: Example of radar-based self-speed estimation compared against the ground truth indicator.

The full process of ego-velocity estimation is summarized in Figure 36. The radar point cloud is first filtered using

RANSAC to retain only inliers corresponding to static structures. From these points, both the azimuth angle and Doppler velocity are extracted. At this stage, the pipeline branches into two estimation strategies:

- **Scalar ego-speed estimation:** a polynomial regression approximates the cosine law of Doppler velocities, yielding the forward velocity component only.
- **2D ego-velocity estimation:** a linear least-squares solution of the Doppler equation

$$v_r(\theta) = v_x \cos(\theta) + v_y \sin(\theta)$$

provides both longitudinal (v_y) and lateral (v_x) velocity components simultaneously.

Both outputs are subsequently passed through a Kalman filter for temporal smoothing before being used in the downstream pipeline.

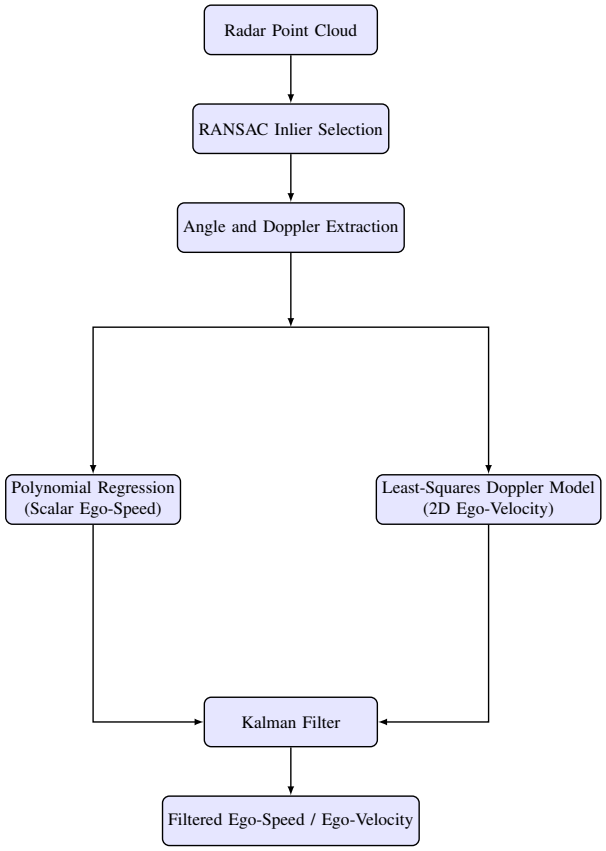


Fig. 36: Block diagram of the self-speed estimation process, showing the two complementary approaches: scalar polynomial fitting and 2D Doppler-based least-squares estimation.

F. Kalman Filter

The Kalman filter is a widely used mathematical algorithm to improve the accuracy of measurements by reducing the noise and uncertainty inherent in sensor data. This is done via estimations of what could be the next state using prior measurements to obtain this "next" state. To enhance the accuracy of the self-speed estimation, a 1D Kalman (meaning that it is used to estimate only one state) filter was implemented. This filter estimates the vehicle's exact velocity based on the provided radar-based seld speed estimation data.

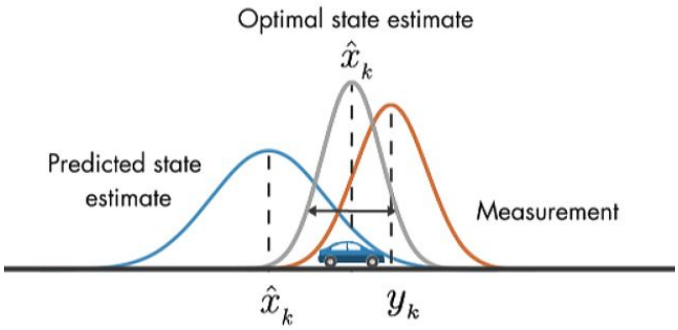


Fig. 37: Kalman filter prediction of the state using estimations. From: [16].

The Kalman filter follows a classical predict-update model, focusing on a single state variable: the vehicle's velocity. The process and measurement uncertainty are encapsulated in the following variables:

- Process variance Q: models the uncertainty in the vehicle's motion (e.g., acceleration changes, jitter).
- Measurement variance R: represents the uncertainty in each self-speed estimation sample.

Where in real life the implementation can be represented as:

- Estimated value: the current velocity estimate.
- Estimated error: the uncertainty (variance) of the current estimate.

At each new frame, a new self-speed estimation is used to update the Kalman filter which effects its output by a variable which is called Kalman Gain 'K':

$$K = \frac{\hat{P}}{\hat{P} + R} \tag{3}$$

Where:

- $\hat{P}$: predicted error variance
- R: measurement variance

This process allows the filter to balance trust between the incoming measurement and the current estimate. Over time, the filter becomes more confident, reducing the influence of noisy measurements.

Thus, the incorporation of the Kalman filter in this project provides smoother, more reliable self-speed estimation, which enhances the performance of the following dynamic filtering stage.

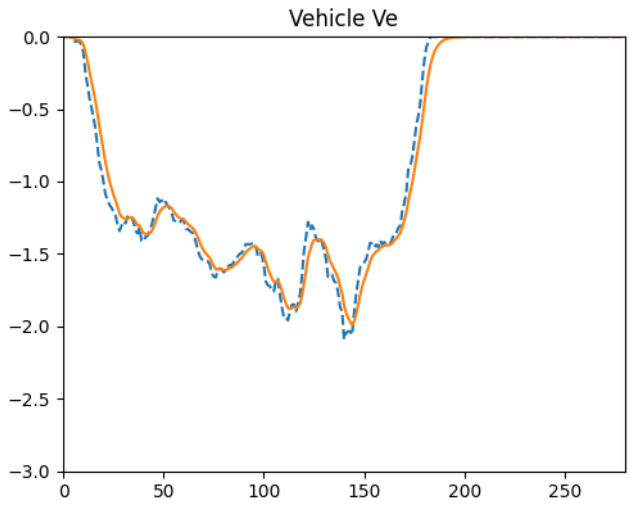


Fig. 38: Raw self-speed vs. self-speed after Kalman filter.

G. Clustering

Clustering is a key technique for grouping similar data points based on their spatial or statistical characteristics. In automotive sensing, clustering enables the detection and tracking of relevant objects such as vehicles, pedestrians, and obstacles by structuring raw radar detections into meaningful groups.

Among commonly used clustering algorithms, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) stands out due to its robustness in handling irregular cluster shapes and its ability to automatically identify noise points without requiring a predefined number of clusters [15]. This contrasts with centroid-based methods such as K-Means, which require specifying K in advance and struggle with irregularly shaped clusters, and with hierarchical methods like agglomerative clustering, which can capture hierarchical structures but are computationally expensive and sensitive to noise.

Algorithm	Type	Strengths	Weaknesses	Best Use Case
DBSCAN	Density-Based	• Detects clusters of varying shapes and sizes • Identifies outliers (noise points)	• Computationally expensive for large datasets	Radar object detection with dynamic object counts and noise filtering
K-Means	Centroid-Based	• Fast and efficient for large datasets	• Requires fixed number of clusters (K) • Poor performance with irregular shapes	Structured radar data with known object counts
Agglomerative	Hierarchical	• No prior knowledge of cluster number required • Can detect hierarchical structures	• Sensitive to noise • High computational cost	Offline grouping of radar signatures

TABLE III: Comparison of clustering algorithms.

DBSCAN defines a cluster based on two parameters: a neighborhood radius ε and a minimum number of points MinPts. For a given point p_i with coordinates (x_i, y_i), its ε-neighborhood is defined as

$$\mathcal{N}_\varepsilon(p_i) = \{p_j \mid \|p_j - p_i\|_2 \leq \varepsilon\}. \tag{4}$$

A point p_i is called a *core point* if

$$|\mathcal{N}_\varepsilon(p_i)| \geq \text{MinPts}. \tag{5}$$

Clusters are then formed by connecting density-reachable points, while points that do not belong to any cluster are classified as noise. This density-based formulation makes DBSCAN well-suited for radar data, where detections of the same object tend to be locally dense, while clutter appears as isolated points.

Two-Stage DBSCAN Rationale: In practice, a single parameter set for $(\varepsilon, \text{MinPts})$ is insufficient because:

- A large ε merges nearby objects into one cluster.
- A small ε causes fragmentation or discards weak reflections.

To address this, a two-stage DBSCAN process was implemented:

a) Stage 1: Permissive Filtering:

$$\varepsilon_1 = 2\,\text{m}, \quad \text{MinPts}_1 = 2 \tag{6}$$

This stage acts as a noise filter by discarding points that cannot form even small local clusters. Outliers are eliminated early, reducing computational load for the next stage.

b) Stage 2: Fine Clustering:

$$\varepsilon_2 = 1\,\text{m}, \quad \text{MinPts}_2 = 4 \tag{7}$$

This stage applies stricter parameters to avoid merging distinct objects into a single cluster. It ensures compact clusters that more accurately reflect real objects in the environment.

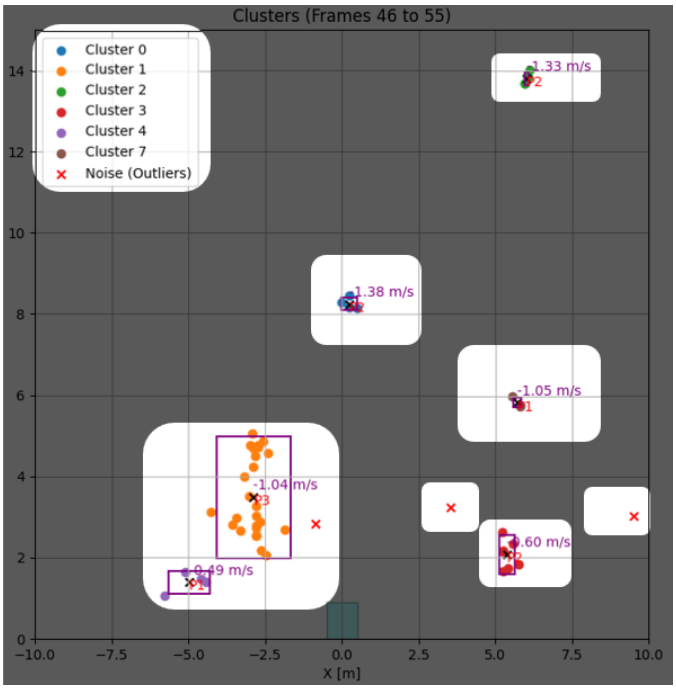


Fig. 39: Illustration of the first clustering stage: isolated outliers are discarded, and only candidate clusters remain.

The resulting process can be expressed as a composition of the two operators:

$$\mathcal{C}_{\text{final}} = \text{DBSCAN}(\varepsilon_2, \text{MinPts}_2,\ \text{DBSCAN}(\varepsilon_1, \text{MinPts}_1,\ P)), \tag{8}$$

where P denotes the raw radar point cloud and $\mathcal{C}_{\text{final}}$ the final set of clusters.

This formulation highlights the dual role of DBSCAN: first as a noise filter, then as a clustering method. By structuring the problem in two passes, the algorithm maintains robustness to noise while achieving finer object separation in dense scenarios.

H. Cluster Tracking

Once clusters have been identified, it is necessary to track them over consecutive frames to ensure temporal consistency. A cluster tracker assigns each detected cluster a unique identifier (ID) and maintains a history of its trajectory over time. This enables the system to distinguish between persistent static objects and transient detections caused by noise or dynamic obstacles.

Hits and Misses: Each tracked cluster is updated at every frame using two indices:

- **Hit Count (h):** The number of consecutive frames in which the cluster has been successfully matched to a detection.
- **Miss Count (m):** The number of consecutive frames in which the cluster has failed to find a corresponding detection.

A cluster is considered *active* if $m \leq M_{\text{max}}$, where M_{max} is the maximum allowed misses before deletion. This mechanism ensures that short occlusions or temporary sensor noise do not cause immediate loss of tracks.

Geometric Association: Let $c_t = (x_t, y_t)$ denote the centroid of a cluster at frame t. For each existing track, association with a new detection is performed by minimizing the spatial distance:

$$d(c_t, c_{t+1}) = \|c_t - c_{t+1}\|_2. \tag{9}$$

If a fitted motion model is available, the distance is instead evaluated relative to the predicted cluster trajectory. In this project, a simple line-fitting approach was applied to the history of each cluster's centroids:

$$y = mx + b, \tag{10}$$

where (m, b) are obtained via least-squares regression over the last N centroids. The orthogonal distance of a new detection (x_0, y_0) to this fitted line is computed as:

$$d_{\perp}(x_0, y_0) = \frac{|mx_0 - y_0 + b|}{\sqrt{m^2 + 1}}. \tag{11}$$

The detection is associated with the track if $d_{\perp} \leq d_{\text{max}}$, where d_{max} is a configurable threshold.

Track Management: The complete tracking procedure consists of:

1) **Initialization:** If no tracks exist, each cluster spawns a new track with a unique ID.
2) **Association:** Existing tracks are matched to current detections using the minimum distance criterion.
3) **Update:** Matched tracks increment their hit count ($h \leftarrow h + 1$), reset their miss count, and append the centroid to their history.
4) **New Track Creation:** Unmatched detections start new tracks with fresh IDs.
5) **Deletion:** Tracks exceeding M_{max} consecutive misses are deleted.

This process ensures that each stationary object is consistently represented by the same track ID across frames, while spurious detections are naturally filtered out.

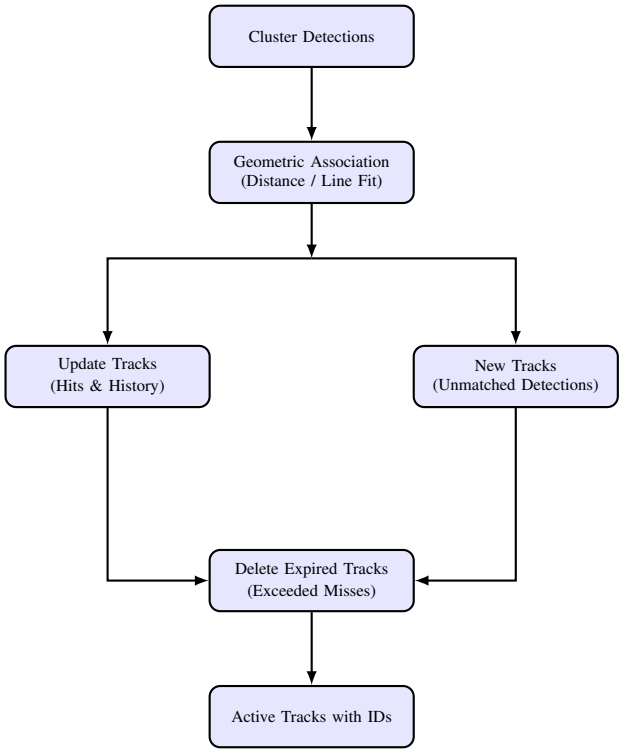


Fig. 40: Cluster tracking block diagram. Each detection is associated with existing tracks or used to initialize new tracks. Tracks are deleted after exceeding miss thresholds.

To illustrate the method, Figure 41 shows a clustered radar point cloud without temporal tracking. Figure **??** shows the same scene after applying the tracker: each cluster is assigned an ID, along with Doppler, centroid coordinates, and hit count. The tracker maintains these IDs consistently across frames, enabling reliable identification of stationary obstacles.

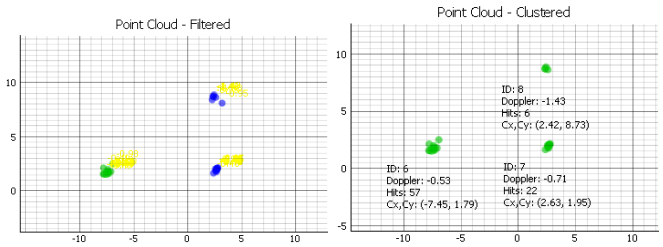


Fig. 41: Illustration of the cluster tracking process. Left: clustered point cloud without tracking. Right: tracked clusters with assigned IDs, Doppler values, centroids, and hit counts.

I. Iterative Closest Point (ICP)

The Iterative Closest Point (ICP) algorithm is a classical method for estimating rigid-body transformations between two point sets. In the context of radar odometry, ICP provides a way to align consecutive frames of radar point clouds in order to estimate ego-motion. The main goal is to determine the relative rotation $R \in SO(2)$ and translation $t \in \mathbb{R}^2$ that minimize the alignment error between a source point cloud $P = \{p_i\}_{i=1}^{N}$ and a target point cloud $Q = \{q_j\}_{j=1}^{M}$.

Mathematical Formulation: The ICP algorithm can be separated into two main steps: correspondence search and transformation estimation.

a) Correspondence Search.: For each point $p_i \in P$, the closest point in Q is selected according to the Euclidean distance:

$$q_i^* = \arg\min_{q_j \in Q} \|p_i - q_j\|_2. \tag{12}$$

This step establishes a set of tentative correspondences $\{(p_i, q_i^*)\}$ between the two frames. Efficient implementations use spatial data structures such as KD-trees to accelerate nearest-neighbor searches.

b) Transformation Estimation.: Once correspondences are found, the rigid-body transformation (R, t) is estimated by minimizing the least-squares error:

$$E(R, t) = \sum_{i=1}^{N} \|p_i - (Rq_i^* + t)\|^2. \tag{13}$$

The optimal solution can be obtained using singular value decomposition (SVD) of the cross-covariance matrix between the two point sets. In 2D, the rotation can be parameterized by an angle θ, such that:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{14}$$

The transformation is then applied to Q, and the process is repeated iteratively until convergence, typically when changes fall below a threshold or a maximum number of iterations is reached.

Global ICP: In the global variant, all available points of the radar point cloud are used to compute correspondences and estimate the motion. This maximizes the use of sensor information but can be sensitive to noise and dynamic objects, since outliers directly affect the transformation estimate.

Cluster-wise ICP: To increase robustness, a cluster-wise approach was also investigated. Instead of using the entire point cloud, correspondences are restricted to clusters identified in the previous stage (see Section V-H). Among these clusters, priority is given to those with the highest *hit count* and lowest *miss count*, since they are more likely to represent stable, stationary objects. For each cluster C_k, an independent ICP alignment is performed:

$$E_k(R, t) = \sum_{p_i \in P_k} \|p_i - (Rq_i^* + t)\|^2, \tag{15}$$

where P_k and Q_k denote corresponding cluster subsets across two consecutive frames. The resulting transformations are then aggregated, typically via weighted averaging based on cluster stability, to obtain the final ego-motion estimate.

Practical Considerations: Several practical aspects influence ICP performance:

- **Initialization:** ICP assumes small inter-frame motion. Poor initialization may cause convergence to local minima.
- **Nearest-neighbor search:** KD-trees enable efficient correspondence search in $\mathcal{O}(N \log N)$ time.
- **Outlier rejection:** Correspondences with distances exceeding a threshold $d_{\max}$ are discarded to reduce the influence of spurious points.
- **Cluster selection:** Restricting ICP to stable clusters improves robustness against moving objects and noise.

J. Iterative Closest Point (ICP) for Ego-Motion Estimation

The Iterative Closest Point (ICP) algorithm estimates the rigid-body transformation between two consecutive radar frames. Given two point sets

$$P = \{p_i \in \mathbb{R}^2\}_{i=1}^{N}, \quad Q = \{q_j \in \mathbb{R}^2\}_{j=1}^{M},$$

the goal is to find a rotation $R \in SO(2)$ and translation $t \in \mathbb{R}^2$ that minimize the alignment error

$$\min_{R,t} \sum_{i=1}^{N} \|Rp_i + t - q_{\pi(i)}\|^2,$$

where $\pi(i)$ denotes the index of the nearest neighbor of p_i in Q.

Nearest Neighbor Matching: To associate points between frames, a KD-tree was constructed over the set Q. For each point $p_i \in P$, its closest match $q_{\pi(i)}$ is obtained as:

$$\pi(i) = \arg\min_j \|p_i - q_j\|_2.$$

Per-Point Motion Estimates: For each matched pair $(p_i, q_{\pi(i)})$, the incremental translation and orientation change are computed as

$$t_i = p_i - q_{\pi(i)}, \qquad \theta_i = \arctan 2\big((p_i - q_{\pi(i)})_y, (p_i - q_{\pi(i)})_x\big).$$

Global Averaging: The global motion is obtained by averaging over all matched pairs:

$$\bar{t} = \frac{1}{N}\sum_{i=1}^{N} t_i, \qquad \bar{\theta} = \frac{1}{N}\sum_{i=1}^{N} \theta_i.$$

Homogeneous Transformation Matrices: The estimated transformation from ICP is expressed in homogeneous coordinates as:

$$T_{\text{ICP}} = \begin{bmatrix} \cos\bar{\theta} & -\sin\bar{\theta} & \bar{t}_x \\ \sin\bar{\theta} & \cos\bar{\theta} & \bar{t}_y \\ 0 & 0 & 1 \end{bmatrix}.$$

For ego-motion estimation (inverse motion), the corresponding matrix is:

$$T_{\text{ego}} = \begin{bmatrix} \cos\bar{\theta} & \sin\bar{\theta} & -(\bar{t}_x\cos\bar{\theta} + \bar{t}_y\sin\bar{\theta}) \\ -\sin\bar{\theta} & \cos\bar{\theta} & \bar{t}_x\sin\bar{\theta} - \bar{t}_y\cos\bar{\theta} \\ 0 & 0 & 1 \end{bmatrix}.$$

Cluster-wise ICP: In addition to the global ICP (using all points), a cluster-wise ICP is applied. Each cluster C_k maintains counters for *hits* and *misses*, ensuring stable targets are prioritized. For each cluster, the transformation (t_k, θ_k) is estimated using the above procedure. The global motion estimate is then computed as a weighted average:

$$\bar{t} = \frac{\sum_k w_k t_k}{\sum_k w_k}, \qquad \bar{\theta} = \frac{\sum_k w_k \theta_k}{\sum_k w_k},$$

with weights

$$w_k = \max(\min(\text{hits}_k^{\alpha},\ \text{max\_hits}),\, 1),$$

where $\alpha \in [1, 2]$ controls the influence of stable clusters.

Python-style Conceptual Flow:

1) Input two frames P and Q.
2) Use a KD-tree to find nearest neighbors between P and Q.
3) For each matched pair $(p_i, q_{\pi(i)})$, compute translation (t_x, t_y) and rotation θ.
4) Average all translations and rotations to obtain $(\bar{t}, \bar{\theta})$.
5) Build the homogeneous transformation T_{ICP}.
6) Derive the inverse ego-motion matrix T_{ego}.
7) (**Cluster ICP**) Repeat steps 2–6 on each cluster and fuse results using hit-based weighting.

VI. Summary and Outlook

Insert conclusion here

References

[1] Segway Inc., "Ninebot Go-kart PRO product page", 2025, Webpage. [Online]. Available: https://de-de.segway.com/products/ninebot-gokart-pro

[2] Texas Instruments, "IWR1642: difference between AWR and IWR parts", 2025, Webpage. [Online]. Available: https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/742730/iwr1642-difference-between-awr-and-iwr-parts

[3] Texas Instruments, "IWR6843AOPEVM product page", 2025, Webpage. [Online]. Available: https://www.ti.com/tool/IWR6843AOPEVM

[4] Texas Instruments, "User's Guide mmWave Demo Visualizer," 2020, Online Document. [Online]. Available: https://www.ti.com/lit/ug/swru529c/swru529c.pdf?ts=1742817596204.

[5] Texas Instruments, "Understanding UART Data Output Format", 2025, Webpage. [Online]. Available: https://dev.ti.com/tirex/content/radar_toolbox_2_20_00_05/docs/software_guides/Understanding_UART_Data_Output_Format.html

[6] Texas Instruments, "mmWave Sensing Estimator", 2025, Webpage. [Online]. Available: https://dev.ti.com/gallery/view/mmwave/mmWaveSensingEstimator/ver/2.5.1/

[7] ub4raf, "Ninebot-PROTOCOL", 2025, GitHub Repository. [Online]. Available: https://github.com/ub4raf/Ninebot-PROTOCOL

[8] -, "Ninebot ES Communicaton Protocol", 2019, Webpage. [Online]. Available: https://cloud.scooterhacking.org/release/nbdoc.pdf

[9] -, "numpy.polyfit documentation", 2025, Webpage. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

[10] Ç. Önen, A. Pandharipande, G. Joseph, and N. J. Myers, "Occupancy Grid Mapping for Automotive Driving Exploiting Clustered Sparsity," *IEEE Sensors Journal*, vol. 24, no. 7, pp. 9240-9250, 2024. [Online]. Available: https://doi.org/10.1109/JSEN.2023.3342463.

[11] D. Casado Herraez, M. Zeller, L. Chang, I. Vizzo, M. Heidingsfeld, and C. Stachniss, "Radar-Only Odometry and Mapping for Autonomous Vehicles," *arXiv preprint*, 2023. [Online]. Available: https://arxiv.org/abs/2305.12409

[12] S. R. Bhatt, B. S. Nadiger, R. Parthasarathy, and H. M. Shetty, "Instantaneous Ego-motion Estimation Using Doppler Radar," *IEEE Sensors Letters*, vol. 7, no. 5, pp. 1–4, 2023. [Online]. Available: https://doi.org/10.1109/LSENS.2023.3244030

[13] C. E. Beal, T. Williams, J. Pauli, M. Mukadam, and B. Boots, "Robust Off-Road Autonomy Using Multimodal Sensor Fusion," in *Proc. of the Conference on Robot Learning (CoRL)*, 2023. [Online]. Available: https://openreview.net/forum?id=kmiZqSgoAt

[14] B. Sundaralingam, C. E. Beal, and B. Boots, "Robust High-Speed State Estimation for Off-Road Autonomous Vehicles," in *Proc. of Robotics: Science and Systems (RSS)*, 2023. [Online]. Available: https://openreview.net/forum?id=3JpFLY3ihix

[15] GeeksforGeeks, *DBSCAN Clustering in ML — Density Based Clustering*, 2023. [Online]. Available: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/. [Accessed: 19-Mar-2025].

[16] MathWorks, *Understanding Kalman Filters, Part 3: Optimal State Estimator*, 2017. Available at: https://la.mathworks.com/videos/understanding-kalman-filters-part-3-optimal-state-estimator--1490710645421.html (Accessed: March 23, 2025).

[17] Texas Instruments, *Radar Toolbox – mmWave Sensor Configuration and Demos*, 2024. Available at: https://dev.ti.com/tirex/explore/node?node=A__ADnbI7zK9bSRgZqeAxprvQ__radar_toolbox__1AslXXD__2.20.00.05 (Accessed: March 23, 2025).