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 single mmWave radar sensor which is mounted in front of the vehicle. This to avoid extra hardware costs and complexity.

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.

***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.

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 velocity. 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.

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.

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.

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. 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.

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.

The experimental setup is illustrated in Figure 1, where 2 mmWave radar sensors are mounted at the front of a vehicle, providing overlapping fields of view to enhance environmental perception. The dual sensor congfiguration improves the spatial coverage, reducing the blind spots and providing a more dense amount of information about the environment for a more stable processing for odometry compared to a single sensor configuration. Each sensor provides a point cloud with range, angle, and Doppler velocity information. Both perspectives are processed by each sensor independently and then merge into a single point cloud for further processing. Creating a more robust and reliable input for the odometry estimation pipeline, enabling the evaluation of radar-based odometry performance in realistic driving scenarios.

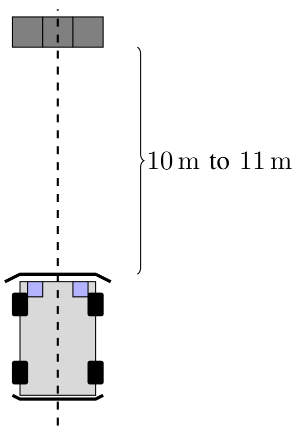


Fig. 1: Test scenario with dual front-mounted mmWave radar sensors.

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, update rate, and data density.
- Selecting sensor configurations that balance 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.
- 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.

III. 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.

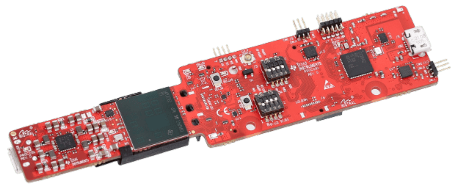


Fig. 2: IWR6843AOP sensor

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].

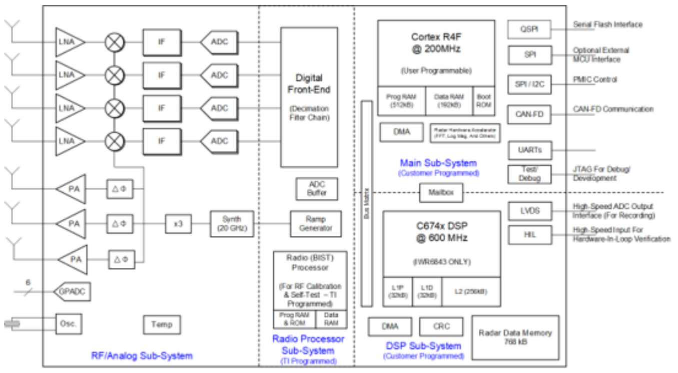

LNA
IF
ADC
PA
ΔΦ
x3
Synth (20 GHz)
GPADC
Osc.
Temp
RF/Analog Sub-System
Digital Front-End (Decimation Filter Chain)
ADC Buffer
Ramp Generator
Radio (BIST) Processor (For RF Calibration & Self-Test – TI Programmed)
Prog RAM & ROM
Data RAM
Radio Processor Sub-System (TI Programmed)
Bus Matrix
Cortex R4F @ 200MHz (User Programmable)
Prog RAM (512kB)
Data RAM (192kB)
Boot ROM
DMA
Radar Hardware Accelerator (FFT, Log Mag, And Others)
Main Sub-System (Customer Programmed)
Mailbox
C674x DSP @ 600 MHz (IWR6843 ONLY)
L1P (32kB)
L1D (32kB)
L2 (256kB)
DMA
CRC
Radar Data Memory 768 kB
DSP Sub-System (Customer Programmed)
QSPI
SPI
SPI / I2C
CAN-FD
UARTs
Test/ Debug
LVDS
HIL
Serial Flash Interface
Optional External MCU Interface
PMIC Control
CAN-FD Communication
JTAG For Debug/ Development
High-Speed ADC Output Interface (For Recording)
High-Speed Input For Hardware-In-Loop Verification

Fig. 3: 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].

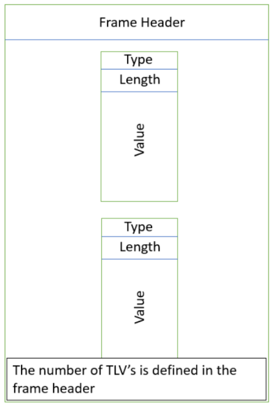
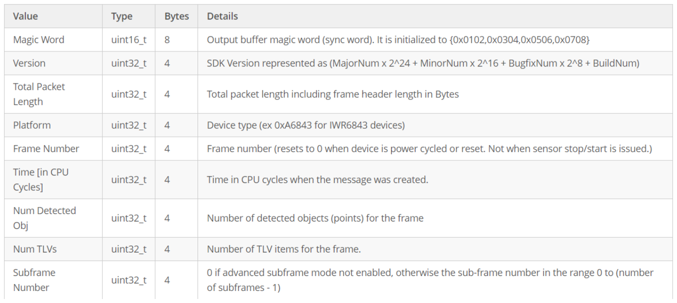


Fig. 4: Frame structure. From [5].

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:

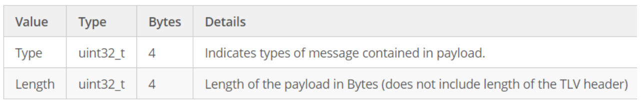
- Magic Word: This value indicates the start of a new header, meaning that it can be used as a starting point for processing each frame.
- Total Packet Length: Total number of bytes in the frame (including the header) which can be used to calculate the frame's end.
- Platform: Indicates the device type 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. 5: Frame header format. From [5].

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. 6: TLV header format. From [5].

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

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 I: Radar System Tuning Parameters and Trade-offs

- 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 an unambigious range of at least $10\,\mathrm{m}$ and a maximum radial velocity of $10\,\mathrm{m\,s^{-1}}$ due to its boundary conditions, together with the desire of the highest possible resolution of the radial velocity to ensure a reliable operation of the radar-based self-speed estimation. Tuning yielded a configuration with the following operating parameters:

- Frame rate: $30\,\mathrm{f\,s^{-1}}$
- Range resolution: $0.178\,\mathrm{m}$
- Maximum unambigious range: $18.22\,\mathrm{m}$
- Maximum radial velocity: $10.24\,\mathrm{m\,s^{-1}}$
- Radial velocity resolution: $0.16\,\mathrm{m\,s^{-1}}$

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. 8.

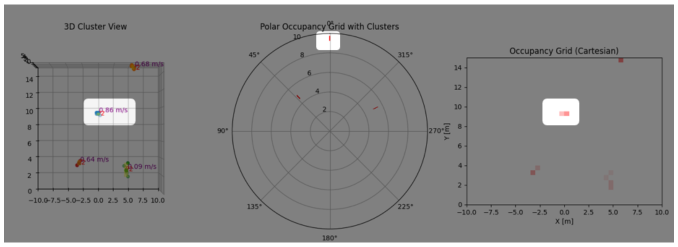


Fig. 7: Output of the IWR6843AOP prior sensor tuning.

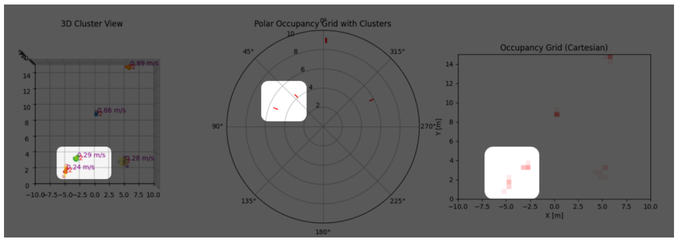


Fig. 8: Output of the IWR6843AOP after sensor tuning.

IV. 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 9, the reflected wave varies in phase and frequency depending on object motion.

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 $B = S \cdot T_c$

Fig. 9: Radar 60-64GHz radio wave reflection.

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:**
$$\Delta v = \frac{\lambda}{2N_cT_c}$$
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.

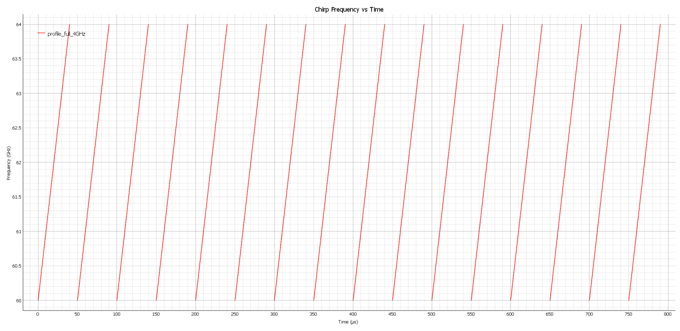


Fig. 10: Single 4 GHz chirp.

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.

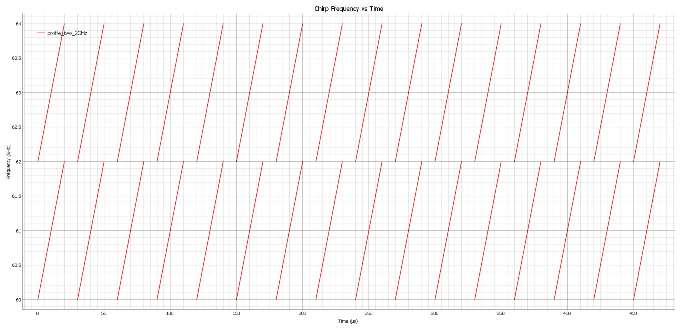


Fig. 11: Two 2 GHz chirps in the same frame.

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

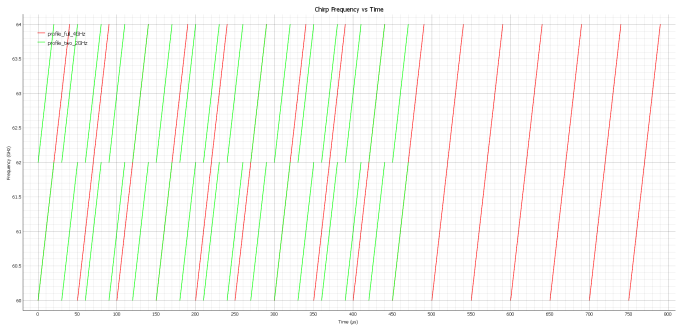


Fig. 12: Overlay of single 4 GHz chirp and two 2 GHz chirps.

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

TABLE II: 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.

C. Summary

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.

V. Pipeline Implementation: Modules Implemented and Mathematical Explanation

To reliably interpret radar sensor data for dynamic object detection and ego-motion estimation, a modular processing pipeline was developed. This pipeline supports both single-radar and dual-radar configurations, and includes integration with an Inertial Measurement Unit (IMU) 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 single-radar setup, 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, and signal-to-noise ratio (SNR). The raw bytes are decoded and structured into frame objects, which are then passed through a Physical Transformation module. This step converts the frame coordinates into the vehicle's local frame, accounting for radar mounting position and orientation. The transformed frames are then passed to the Frame Aggregator. The aggregator buffers several frames, creating a temporal submap that increases point density and improves the stability of downstream operations. This aggregation step is especially important for low frame-rate radars or environments with sparse returns.

Dual-Radar and IMU Integration

In the enhanced dual-radar configuration, two front-facing radar sensors—mounted on the left and right sides—provide overlapping fields of view. Their data streams are synchronized and passed through an IMU module before aggregation. Before being fused in the IMU module, both radar streams undergo an initial Physical Transformation step to account for their respective mounting positions. The IMU plays a critical role in estimating the rotation of the system (typically yaw, θ), which is then used to correctly transform radar point clouds into a common reference frame. This correction is essential when performing operations such as frame aggregation and ego-motion estimation, especially during turning maneuvers or curved trajectories. By incorporating rotation from the IMU, we avoid accumulating drift in the pose estimation and ensure consistent alignment across frames. The rotation is applied as a transformation matrix $R(\theta)$ during the aggregation step.

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:** Compares each point's Doppler-derived velocity v_r against the estimated self-velocity v_e to remove static or inconsistent returns. This step also serves as a soft segmentation between dynamic and static points.

Self-speed estimation is derived from the filtered points and refined using a Kalman filter to stabilize fluctuations. This self-speed value is used both for filtering and for downstream modules.

Robust Clustering and Ego-Motion Estimation

The filtered data is processed using a two-stage clustering algorithm:

- The first stage is permissive, allowing loosely grouped points to form clusters.
- The second stage is strict, refining the clusters to remove noise and assigning each a stable ID.

This approach enables robust tracking of objects over time, even in cluttered or partially occluded environments. To improve robustness against outliers, RANSAC is applied before clustering, ensuring that spurious points do not skew the detection. Finally, ICP (Iterative Closest Point) is used to align point clusters between consecutive frames. Combined with known vehicle motion and IMU-based rotation, this allows for accurate ego-motion estimation and the construction of a consistent world model.

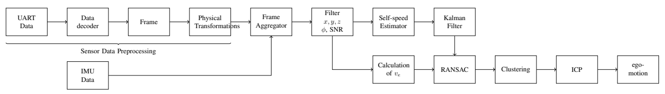


Fig. 13: Block diagram of the pipeline

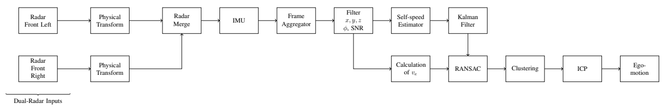


Fig. 14: Dual Radar Pipeline with Physical Transformations and IMU Integration

A. 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.

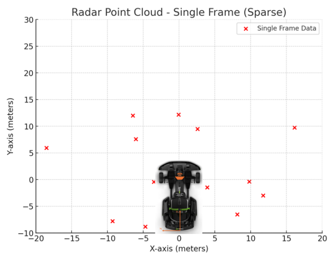


Fig. 15: Single Frame visualization.

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 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:

$$\frac{\sigma^2}{N} \tag{1}$$

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.

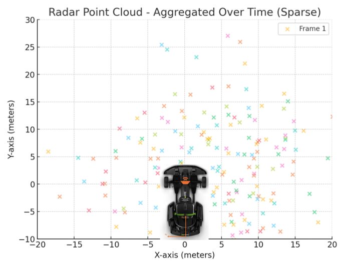


Fig. 16: Frame Aggregation visualization.

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.

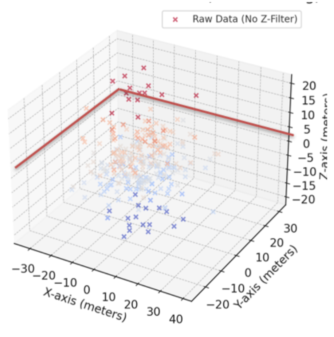
B. "x, y, z, SNR" Filter

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:

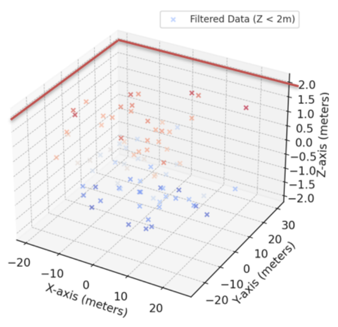
1) Filtering by SNR: All points with a SNR lower than $12\,\mathrm{dB}$ are filtered out to remove points with a low signal and those that might be caused by noise or clutter.
2) Filtering by z coordinate: All points below a z value of $0\,\mathrm{m}$ and above $2\,\mathrm{m}$ are filtered out to remove 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. 17: Example visualization of the input and output when filtering by the value of the z coordinate.

C. 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 because really handy, as it will do the grouping for us, by ussing the radial speed.

D. Self-speed Estimator

The vehicle's self-speed is used for distinguishing between points of stationary and moving objects and filtering out the points of moving objects in a later block 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:

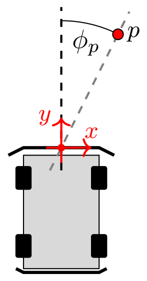


Fig. 18: Definition of the angle ϕ_p

These angles ϕ_p can be calculated using the points' cartesian position information:

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

A curve $v(\phi)$ is fitted through the points with their angles ϕ_p serving as the independent variable and their radial speeds $v_{r,p}$ as the dependent variable. The curve's value $v(\phi = 0°)$ then gives an estimation for the vehicle's self speed.

This is possible because the perceived radial speed $v_{r,p}$ of a point related to a stationary target is dependent on the angle

ϕ_p and follows a cosine when being passed at a certain speed v_0:

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

At $\phi = 0°$, the radial speed is therefore equal to the vehicle's self speed. Because there is rarely a point exactly at $0°$ and the radial speeds of the points also include noise, the approach uses all available points after the first filtering stage to fit a curve and effectively give the best possible estimation. The implementation uses a least squares polynomial regression approach to fit a 2nd order polynomial (see [8]), which is sufficient in this case as the angle ϕ is defined in a way that $\phi_p = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, so the cosine can be approximated by a 2nd order polynomial:

$$cos(\phi_p) \approx a \cdot \phi_p^2 + b \quad with \ \phi_p = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

Multiple runs of this algorithm with recorded test data proved the technique's principle.

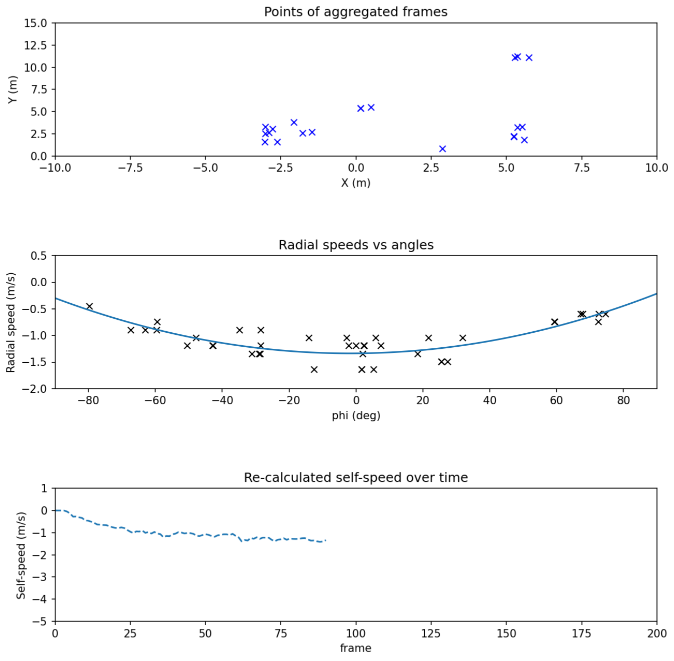


Fig. 19: Test run with recorded data and visualization.

During tests, the self-speed estimated by this technique matched the go-kart's built-in speed indicator with some small static offset but also showed heavy fluctuations from time to time when the radar sensor's output only contained a small amount of points. Although this influence could be reduced by tuning the amount of frames stored in the frame aggregator, a stage containing a Kalman Filter was added after the self-speed estimation before the self-speed value is passed to the following pipeline blocks.

E. 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.

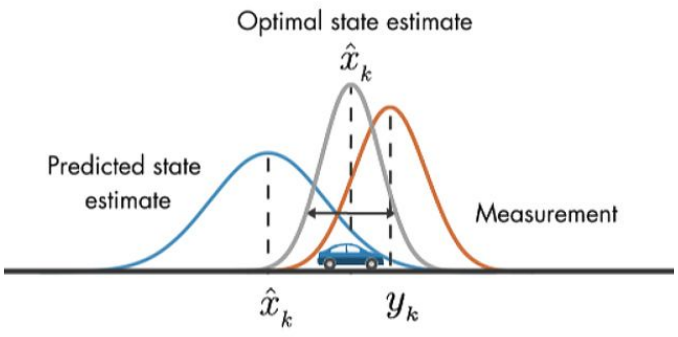


Fig. 20: Kalman filter prediction of the state using estimations. From: [15].

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':

$$K = \frac{\hat{P}}{\hat{P} + R} \tag{2}$$

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.

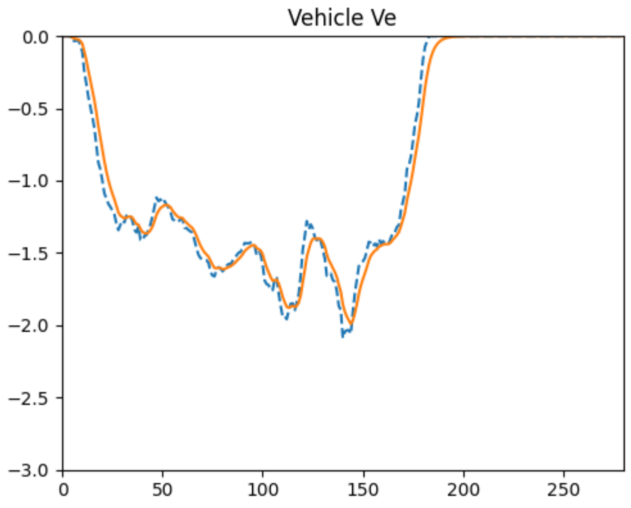


Fig. 21: Raw self-speed vs. self-speed after Kalman filter.

F. *Calculation of v_e and v_e vs. Self-Speed Filter*

As the emergency braking system should only react to stationary targets, only the points of those targets should be passed to the next stage, the clustering stage. Filtering out all points that might be invalid or from moving targets ensures a reliable operation of the clustering process and also effectively reduces processing time as a result of the smaller amount of points. This filtering operation was implemented using a simplistic approach by assuming that theoretically all points of stationary targets should show a velocity equal to the vehicle's velocity when the radar sensor is moving with the vehicle. By comparing the points' velocities to the prior obtained self-speed, invalid points or those of dynamic targets can be separated and filtered out. Due to the working principle of radars and the resulting radar sensor's output format, the dependency between the radial speed and the point's angle to the radar sensor must be taken into account.

The whole block was separated into two sub-stages following after each other. In the first sub-stage, the angle-independent points' speeds, called $v_{e,p}$, are calculated. The following sub-stage executes the filtering operation by comparing the points' $v_{e,p}$ to the vehicle's velocity, supplied by the Kalman filter's output. The calculation of $v_{e,p}$ followed the same approach that was used in the self-speed estimator. An angle ϕ_p was defined between each individual point p of the point cloud and the radar sensor's centerline (see 18). This angle is calculated for every point by using the prior presented formula. The dependency between the point's radial speed $v_{r,p}$ and the calculated angle ϕ_p is then used to calculate each point's angle-independent speed $v_{e,p}$:

$$v_{e,p} = \frac{v_{r,p}}{cos(\phi_p)}$$

A division by zero never occurs because of the first static filtering stage. Filtering by $v_{e,p}$ is done by calculating the difference between the estimated vehicle's self-speed and $v_{e,p}$ and comparing it against a threshold. The comparison of the difference to a threshold is needed as $v_{e,p}$ will rarely be exactly equal to the vehicle's self-speed. A threshold of $0.5\,\mathrm{m\,s^{-1}}$ was found out to be sufficient. After filtering, the points are passed to the clustering stage.

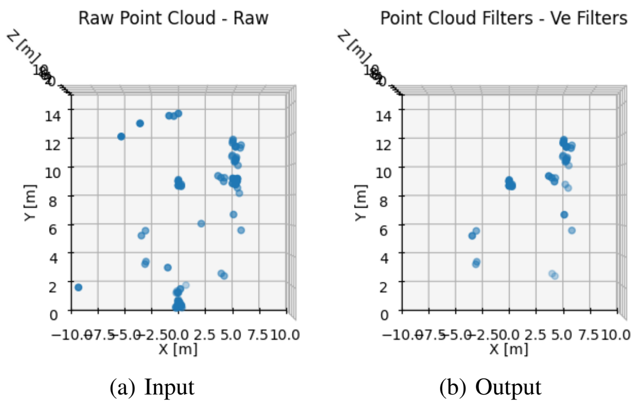


Fig. 22: Example visualization of the input and output data before and after passing the v_e vs. self-speed filtering stage.

G. *Clustering*

Clustering is a fundamental technique used to group similar data points based on their characteristics. It is particularly useful in sensor data analysis for automotive applications, enabling the effective detection and tracking of objects such as vehicles, obstacles, and pedestrians.

Among various clustering algorithms available, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) stands out due to its ability to automatically detect clusters of varying shapes and sizes (see [14]), and to effectively identify and manage noise or outliers. Unlike centroid-based methods such as K-means, which require specifying the number of clusters in advance and struggle with irregularly shaped data, DBSCAN identifies clusters based on the density distribution of data points. Additionally, hierarchical methods like agglomerative clustering, although flexible, tend to be sensitive to noise and computationally expensive for large datasets.

Algorithm	**Type**	**Strengths**	**Weaknesses**	**Best Use Case**
DBSCAN	Density-Based	• Automatically detects clusters of **different shapes and sizes** • Identifies **outliers (noise points)**	• Computationally expensive for **large datasets**	Radar object detection with noise filtering
K-Means	Centroid-Based	• Fast and efficient for **large datasets**	• Requires a **fixed number of clusters (K)** • Struggles with **irregularly shaped clusters**	Segmenting structured radar data with known object counts
Agglomerative	Hierarchical	• Does not require specifying number of clusters • Can be modified to detect **hierarchical structures**	• Can be **sensitive to noise** • Computationally expensive for **large datasets**	Grouping similar radar signatures in post-processing

TABLE III: Comparison of Clustering Algorithms

For this project, DBSCAN's strengths align closely with the requirements of radar object detection, where the number of detected objects can change dynamically and the presence of noise is common. DBSCAN's capability to handle varying densities and noisy data makes it the optimal choice for accurately and efficiently analyzing real-time radar sensor data in automotive applications.

However, this is insufficient to fully meet the requirements of object detection. It is acknowledged that DBSCAN is among the most efficacious and straightforward instruments available for achieving this objective. However, it is important to note that the sizes of the objects under observation can vary significantly from one frame to the next, and the presence of noise can introduce additional variability. To address these challenges, a two-stage clustering approach was chosen, utilizing DBSCAN as a filter and a clustering algorithm. The first stage that acts as a filter with DBSCAN parameterized by an Epsilon of $2\,\mathrm{m}$ and a minimum number of samples to specify a core point of 2. The justification for using it as a filter is that, at first, a broad or permissive set of parameters is applied, making it easier to filter out points or data that are considered as noise. This is not due to the presence of "real" noise; rather, it is owing to a lack of the required amount of points to qualify as an object. This principle is represented in the next figure:

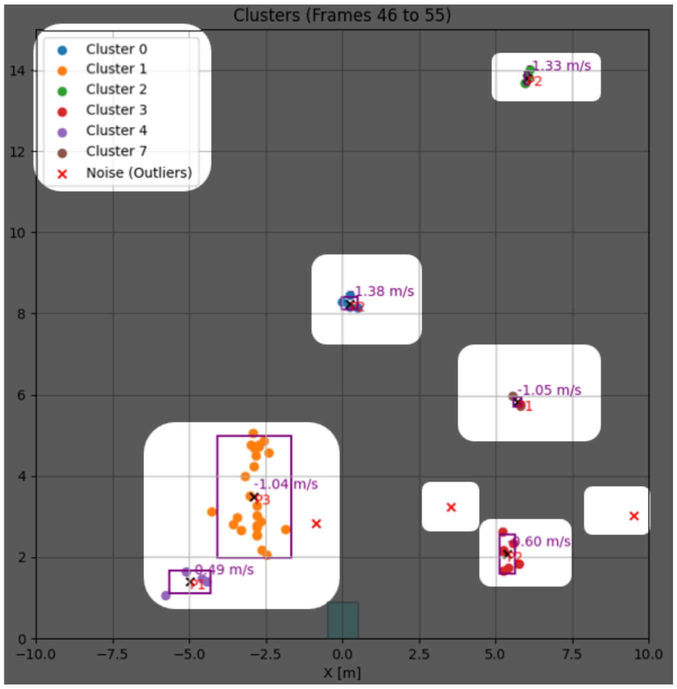


Fig. 23: First clustering stage

It can be seen that there are three points that are not included in the clusters and seem to be outliers. These points might belong to an object, but most likely they are caused by clutter or noise. As they are not included in clusters after first stage, they are discarded and only the points that are included in clusters are passed to the second stage.

The second stage utilizes DBSCAN with a finer set of parameters. In this stage, Epsilon was chosen to be equal to $1\,\mathrm{m}$ and the minimum number of samples to specify a core point was set to 4. This results in finer clusters, preventing objects which are close to each other to be recognized as one single cluster. The clusters are then passed to the brake controller for final object detection and handling of approaching targets.

H. Brake Controller

A brake controller is a crucial component in autonomous and assisted driving systems, enabling vehicles to respond intelligently to detected obstacles or hazards. In this project, a simple yet effective braking mechanism was implemented using a linear controller that calculates a safe stopping distance based on the vehicle's current speed. The controller assumes a linear relationship between speed and stopping distance, a reasonable approximation under controlled conditions and for the rather lightweight go-kart. The controller's objective is to ensure that if the vehicle is approaching an obstacle inside an area of interest and potentially getting into contact with it, it activates the brake early enough to stop within a safe distance. Here, the area of interest is a specified area in front of the go-kart encasing the space where an approaching stationary obstacle could become a hazard to the driver and vehicle. As the distance which is required to stop the vehicle before getting into contact with the obstacle is proportional to the vehicle's speed, the length of the area of interest is coupled to it.

The brake controller's logic is based on the following variables:

- Stopping distance $d_{\text{stop}}(v_{\text{current}})$: Estimates how much distance the vehicle requires to come to a complete stop at the current speed, according to the vehicle specifications.
- Output signal $brake\_signal(d_{\text{stop}}, d_{\text{target}})$: Generates a binary brake signal (0 or 1) based on whether the stopping distance exceeds the target distance. As the current brake controller simply considers if the brake should be applied or not.

The stopping distance is computed relative to a reference speed and stopping distance (default: 40 kph $\rightarrow$ 6 meters), allowing for linear scaling:

$$d_{\text{stop}}(v_{\text{current}}) = \frac{v_{\text{current}}}{v_{\text{ref}}} \cdot d_{\text{ref}}$$

If the required stopping distance is greater than or equal to the available distance (measured distance to a detected obstacle), the system triggers full braking:

$$brake\_signal(d_{\text{stop}}, d_{\text{target}}) = \begin{cases} 1 & \text{if } d_{\text{target}} \leq d_{\text{stop}} \\ 0 & \text{otherwise} \end{cases}$$

This straightforward logic allows the controller to make quick decisions in real time, helping the vehicle to react safely to moving and stationary objects, detected by the radar. Although the system is designed to be cautious because it only switches between braking and not braking, it could be improved later by adding smoother braking or more advanced features like adaptive cruise control. The controller plays a key role in the interaction between detection (from the radar sensor's output data and the processing pipeline) and actuation (braking), forming the basis for a closed-loop safety mechanism.

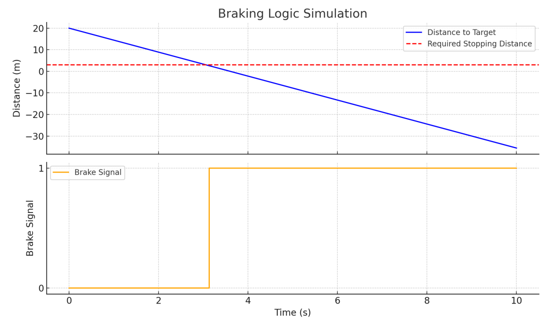


Fig. 24: Braking decision based on stopping distance estimation.

VI. Summary and Outlook

In this project, an object detection system for a consumer-grade electric go-kart was implemented by using a mmWave radar sensor. The system successfully meets its primary objectives by accurately detecting obstacles through a custom point-cloud analysis algorithm. Although the current implementation is limited to stationary objects, it establishes a strong foundation for future development. All necessary components, including a modular processing pipeline and a hardware interface for safely manipulating the go-kart's brake signal, were developed.

The pipeline's modular architecture proved to support a dynamic developing process, which turned out to be necessary when working with point clouds from radar sensors and an electric go-kart that was not intended to be modified by the end-user. It also enables further development by expanding, exchanging or modifying processing stages. During the development, two stages turned out to be extraordinary helpful when it comes to mitigating the influences of the potentially heavily fluctuating point cloud data. The frame aggregator in combination with running the radar sensor at a high frame rate successfully tackled the problem of data sparsity that sometimes occurred in the test scenario environment. This was caused by the scenario's ”clean” setup without a huge number of targets. The two-stage clustering approach using the DBSCAN algorithm proved to be able to reliably filter out outliers caused by clutter or noise and was able to provide the brake controller with stable information on stationary objects.

In addition to those two stages, the usage of multiple filtering stages of static and dynamic behavior proved to support the reliability of the whole system. Static filtering stages early in the pipeline are able to filter out irrelevant points or those with a low level of confidence by using their spatial coordinates and SNR information. This reduces the required processing time of the later stages by condensing the mass of data to the relevant points. The dynamic filtering stage allows a reliable differentiation between points that are caused by stationary and moving targets by leveraging the information of the radar-only self-speed estimation. The approach of using a distance that is linear to the vehicle's velocity to decide whether an emergency braking event should be triggered, and outputting a binary signal, turned out to be sufficient, as the balance board's internal controller prevents the wheels from locking.

As with any project, there is always room for improvement, and this work is no exception. Although the current implementation meets the initial goals and requirements, the algorithm can still be refined. The project is currently implemented in a threaded solution using Python, which is a result of the dynamic development process where a lot of different techniques and approaches where implemented, tested and sometimes discarded. Switching from C++, which was used initially used for development, to Python allowed for a quicker development with simpler possibilities of visualization, but also caused a noticeable lack of performance. This lack of performance showed up in the last stages of development, during testing, and created a delay in the response when tested in an autonomous environment, meaning that when the implementation was not powered by a sufficient power supply, the implemented system showed certain delays in the response when an object was detected in the area for activation of the brake. It is assumed that the delay originates from Python's heavyweight interpreter and should vanish after porting the system's implementation back to C++. As for this reason, the decision of not fully merging the existing system into the test vehicle at this stage of the development was made, as it did not provided a fully safe environment for testing. The validation was done with a LED to indicate the activation of the emergency brake. An improvement or next step would be to migrate the system back to C++, where the threaded implementation would provide a better response time for each running task. Further improvements can also be made by incorporating occupancy grids through a Bayesian filter to enhance object detection, allowing the system to go beyond stationary targets. By improving the system's object detection capabilities to cover moving targets, it would also be possible to estimate their direction of movement, which could significantly improve the driving assistance algorithm and contribute to accident prevention through a more precise analysis.

The present status of this project is available in the following GitHub repository: Radar-mmWave on GitHub.

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

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

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

[9] 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.

[10] D. Casado Herraez, M. Zeller, L. Chang, I. Vizzo, M. Heidingsfeld, and C. Stachniss, “Radar-Only Odometry and Mapping for Autonomous Vehicles,” *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2305.12409.

[11] L. Sless, G. Cohen, B. E. Shlomo, and S. Oron, “Road Scene Understanding by Occupancy Grid Learning from Sparse Radar Clusters using Semantic Segmentation,” in *Proc. ICCV Workshop*, 2019.

[12] Z. Wei, R. Yan, and M. Schreier, “Deep RADAR Inverse Sensor Models for Dynamic Occupancy Grid Maps,” *arXiv preprint*, 2024. [Online]. Available: https://arxiv.org/abs/2305.12409v3.

[13] M. Li, Z. Feng, M. Stolz, M. Kunert, R. Henze, and F. Küçükay, “High Resolution Radar-based Occupancy Grid Mapping and Free Space Detection,” in *Proc. 4th Int. Conf. Vehicle Technol. Intell. Transport Syst. (VEHITS)*, 2018, pp. 70-81.

[14] 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].

[15] 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).

[16] 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).