Development of a Radar-Based Emergency Braking System

Leveraging mmWave Radar Sensor Technology for Object Detection

1st Luis Fernando Rodriguez Gutierrez
Fachhochschule Dortmund
M.Eng. Embedded Systems Engineering
luis.rodriguez001@stud.fh-dortmund.de

2th Leander Hackmann
Fachhochschule Dortmund
M.Eng. Embedded Systems Engineering
leander.hackmann001@stud.fh-dortmund.de

***Abstract*—This paper explores the integration of mmWave radar technology for object detection and environmental perception of an emergency braking system, a topic that has gained significant attention in autonomous systems and Advanced Driver Assistance Systems (ADAS). An implementation of an emergency braking system for an electric go-kart is presented with a focus on the system's conception and how a processing pipeline for cluster-based detection of static objects can be developed.**

The presented approach leverages point-cloud data of a IWR6843AOP mmWave radar sensor from Texas Instruments, processed by a modular pipeline to detect and track objects within a 10-meter range. This modular pipeline features a radar-only self-speed estimation together with the combination of a specialized filtering stage and a two-stage clustering technique, effectively implementing a lightweight and simple framework for detecting static targets from a moving vehicle. Its output is used to trigger an emergency braking event which effects the go-kart's braking system through a hardware interface.

***Index Terms*—mmWave radar, object detection, ADAS, autonomous systems, clustering, physical filtering, two-stage filtering, Doppler velocity estimation, occupancy grid mapping, environmental perception.**

I. Introduction

The advancement of autonomous systems and Advanced Driver Assistance Systems (ADAS) has increased the demand for robust and reliable perception technologies. Among these, mmWave radar sensors have emerged as a promising solution due to their ability to operate effectively under various environmental conditions, such as low visibility, fog, rain, and darkness where traditional vision-based sensors, such as LiDAR or cameras, tend to have difficulties. In those cases, where pure optical sensors do not maintain reliability due to the given operating conditions, mmWave radar sensors can still provide range, velocity, and angle information. They therefore represent an essential component in modern perception frameworks, either as the only sensor that implements the solution or through a sensor-merge solution that compensates for the weaknesses of the various optical sensors.

This paper presents a straightforward approach on how to implement an object detection and emergency braking system for an electric go-kart by utilizing Texas Instrument's IWR6843AOP mmWave radar sensor. The implementation involves a modular processing pipeline composed of several building blocks which are designed for real-time data processing. These components encompass the entire process: from the data-acquisition and initial processing of raw radar point cloud data to the final triggering of emergency braking events.

The approach integrates multiple filtering techniques, radar-based self-speed estimation (estimation of the vehicle's own velocity) together with clustering algorithms. These techniques are applied to effectively differentiate static from moving objects and enable the precise detection and tracking of objects.

In particular, static and dynamic filtering stages are combined with a two-stage clustering block to ensure a reliable detection of static objects. The static filtering techniques use the point cloud's SNR information and spatial coordinates to refine the data in the pipeline's early stages by discarding points that are identified as clutter or those that are simply outside the monitored area. Doppler velocity measurements of the points compared to the vehicle's self-speed enable the differentiation between static and dynamic objects, improving overall tracking capabilities and enabling accurate motion predictions.

II. Objective and Sub-Tasks

The main goal of this project is the development of an emergency braking assistant for an electric go-kart that utilizes a radar sensor together with an embedded computing device to detect approaching stationary obstacles and to trigger an emergency braking event when needed. The test scenario for the emergency braking event consisted of a wall built by 3x3 cubes with a side length of $0.4\,\mathrm{m}$, resulting in a total size of $1.2\,\mathrm{m}$ by $1.2\,\mathrm{m}$. With the wall being approached in a straight line at constant speed from a distance of $10\,\mathrm{m}$ to $11\,\mathrm{m}$, the system should recognize the approaching static object and execute an emergency braking event, effectively stopping the go-kart before it gets into contact with the wall.

A Ninebot Go-kart PRO electric go-kart was selected as the test vehicle for this implementation, equipped with a Texas Instruments IWR6843AOPEVM development board, featuring the IWR6843AOP mmWave radar sensor, for data acquisition and sensing tasks.

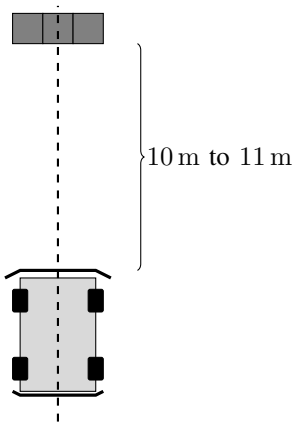


Fig. 1: Test scenario

A. Sub-Tasks

The main objective, together with the given boundary conditions in form of the given test scenario and the provided electric go-kart and radar sensor, implied several practical sub-tasks:

- Finding ways to manipulate the go-kart's controller for performing the braking operation.
- Interfacing the radar sensor and finding a suitable configuration.
- Choosing an embedded computation device.
- Developing and implementing a live processing pipeline.
- Joining the individual parts and testing the system.

As the usage of a generic electric go-kart required its reverse engineering for finding possible ways of interfacing the go-kart's braking system and those findings significantly constrained the solution space of the other sub-tasks, the tasks' ordering was kept and the whole project followed an agile approach.

III. Interfacing the Ninebot Go-kart PRO's Control System

The Ninebot Go-kart PRO is a widely available consumer-grade electric go-kart. It is build in a modular way, extending a standard Ninebot S MAX balance board to a go-kart by adding a frame with a remote controller located inside the cart's steering column which connects to the balance board. As the product was not intended to be modified by the end user and closed-source during the project's development phase, there was no official option to interface with its control system. Reverse engineering was executed for investigating possible ways how the go-cart's brake signal could be manipulated.

A. Architecture of the Go-Kart's Remote Controller

In a first step, the remote controller was taken apart and analyzed. It was found out that it acts as an encapsulated system with an independent power supply in form of batteries which are also located inside the steering column. The system uses a nRF51822 System On Chip (SoC) from Nordic Semiconductor for processing the gas and brake pedals' signals and communicating with the balance board. Both gas and brake pedal utilize a hall sensor to output an analog voltage proportional to the position of the pedal's lever, which is converted from analog to digital domain by using the SoC's internal Analog to Digital Converter (ADC).

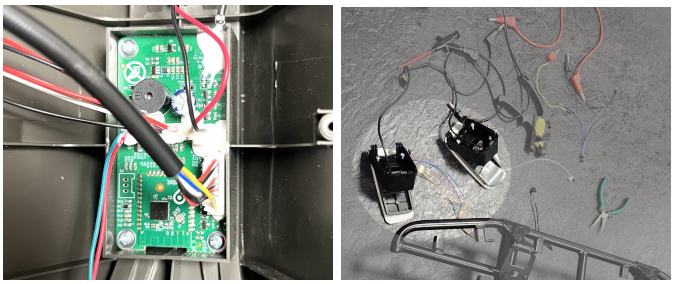


Fig. 2: The remote controller's board inside the steering column and the dismantled brake-pedals.

These signals are passed from the remote controller to the balance board by using two separate communication channels: a wired connection through a cable inside the go-kart's frame and a wireless connection via Bluetooth is used [1]. Multiple sources of documented successfully reverse engineered wireless communication interfaces of older Ninebot scooters stated that a proprietary protocol using a virtual UART at a speed of $115\,200\,\mathrm{Baud}$, without parity-bits and with 1 stop-bit, was used for the wireless communication [6] [7].

Tracing the pins which are used by the wired connection on the remote controller's Printed Circuit Board (PCB) and the labeling on the PCB's backside ("RXD" and "TXD") also suggested a UART connection. These findings implied three possible ways of taking over the control of the brake pedal's signal:

1) Reverse engineering and manipulating the communication of the wired UART connection.
2) Manipulating the analog signal of the brake pedal.
3) Reverse engineering and manipulating the wireless communication protocol of the Bluetooth virtual UART connection.

As manipulating the brake signal via the wired UART connection posed to be the second simplest way while potentially offering more possibilities of getting control of the go-kart or obtaining information from it, compared to the approach of manipulating the analog signal, this way was investigated first.

A Digital Storage Oscilloscope (DSO) with an integrated UART decoder was connected to the data lines to be examined. According to the resources, a $2\,\mathrm{B}$ long frame header with the sequence "$0x5A\ 0xA5$" should have been present at the beginning of each packet [7]. All possible combinations of the UART decoder regarding speed, polarity, parity-bits, and stop-bits were tested but none showed the expected frame header. Additionally, the presence of big voltage spikes with an amplitude of up to $4\,\mathrm{V}$, dependent on the motor's Revolutions Per Minute (RPM), was discovered.

In a second test, it was investigated whether there was a dependency between the frames sent and the value of the gas and brake signal. For this test, the rising edge of the brake pedal's analog signal output was used to trigger the DSO's capture event. The pedal was slowly pressed, kept fully pressed for approximately $175\,\mathrm{ms}$ time and then was released again. This procedure yielded the recording of a communication sequence over a longer period of time. It was carefully checked for dependencies between the brake signal and the frames' contents and the periodicity of communication events, but there was no evidence of correlation.

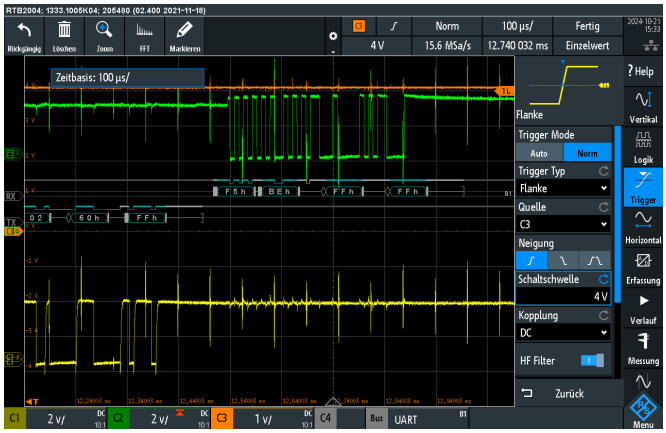


Fig. 3: Plot of the communication between the remote controller and the balance board in idle. Voltage spikes of up to $4\,\mathrm{V}$ can be observed.

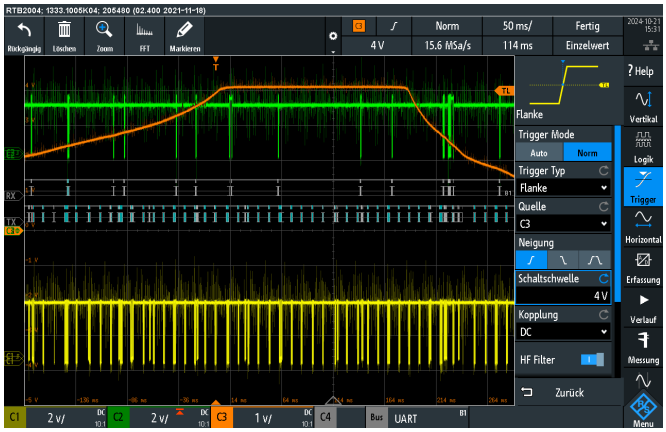


Fig. 4: Plot of the communication between the remote controller and the balance board while using the brake pedal's analog signal as the trigger.

Because no source explicitly stated that the wired connection was used for exchanging gas and brake signals, two last tests were executed. In the first test, the go-kart was powered up with the wired connection between the remote controller and the balance board being detached beforehand. The go-kart refused to enter its normal operating mode and stayed in a "safety-mode" until power was cycled with the wired connection re-attached. In the second test, the go-kart was powered up normally and the wired connection was detached while it was already in its normal operating mode. The go-kart continued to operate normally, fully relying on the wireless communication for the transmission of gas and brake signals. The results of these tests led to the assumption that the wired connection is not used for the transmission of gas and brake signals or only as a fallback option if the wireless communication is disrupted during normal operation. Since deliberately blocking the wireless connection was not possible and fully replacing the remote controller not an option, the manipulation of the brake signal was performed by overwriting the pedal's analog voltage.

B. Interfacing the Brake Pedal

The brake pedal outputs a voltage in the range of $1.2\,\mathrm{V}$ to $4.2\,\mathrm{V}$ which is proportional to the position of the pedal's lever (see 4, orange plot). As the brake pedal's signal has priority over the gas pedal's signal (the go-kart stops if both pedals are pressed simultaneously), it is sufficient to overwrite its signal by the emergency braking system in case of an emergency braking event. Overwriting can be done by adding the two signals together and feeding the result into the remote controller's input. The signal needs to be clipped at the maximum input level of $4.2\,\mathrm{V}$, to protect the remote SoC's analog inputs from overvoltage. A simplistic design incorporating three operational amplifiers was developed for implementing these stages:

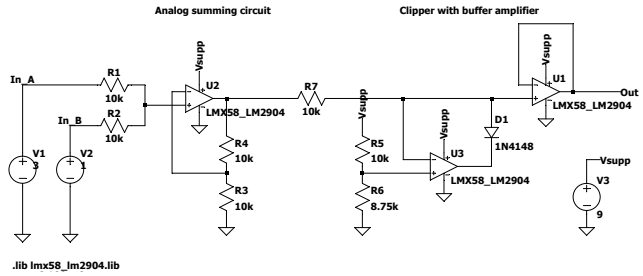


Fig. 5: Analog summing circuit followed by a clipper and a buffer amplifier

A buffer amplifier (voltage follower) was added to lower the circuit's output impedance and to mitigate the influence of the connected load on the clipping voltage. The insertion of this circuit between the brake pedal's output and the remote controller's input allows the emergency braking system to overwrite the brake signal to initiate a braking event while keeping the pedal's normal functionality.

IV. Interfacing the IWR6843 mmWave Radar Sensor

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.

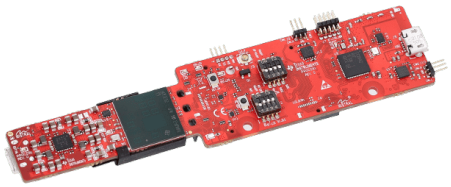


Fig. 6: IWR6843AOP sensor

The IWR6843AOP radar sensor operates within the frequency range of $60\,\mathrm{GHz}$ to $64\,\mathrm{GHz}$ 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].

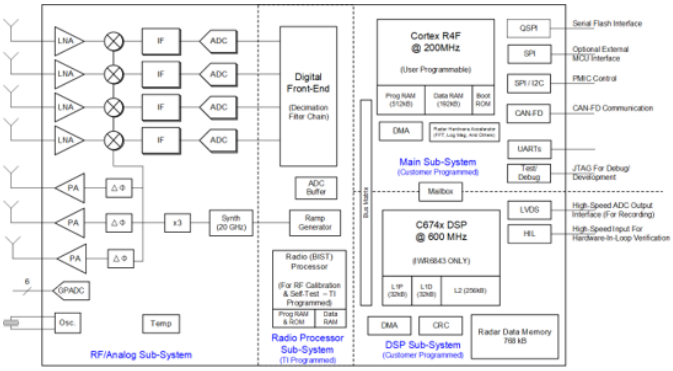


Fig. 7: IWR6843AOP internal Block Diagram

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 onboard 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].

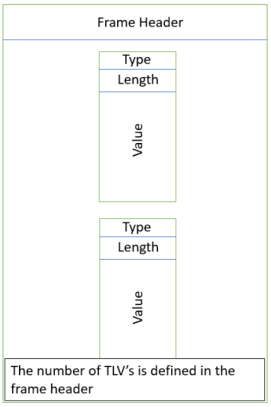


Fig. 8: Frame structure. From [5].

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. 9: 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. 10: 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

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

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 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 yieled 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. 12.

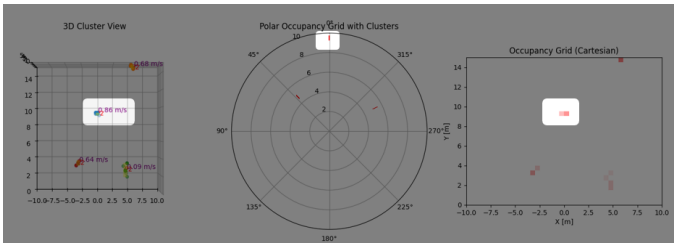


Fig. 11: Output of the IWR6843AOP prior sensor tuning.

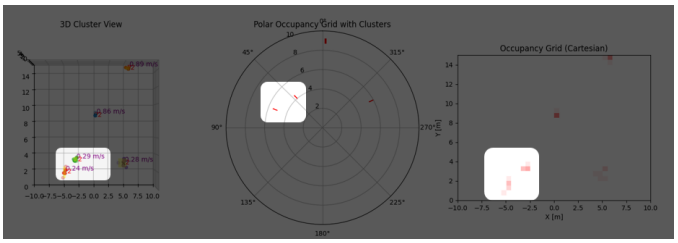


Fig. 12: Output of the IWR6843AOP after sensor tuning.

V. Pipeline implementation: Modules implemented and mathematical explanation

As the radar sensor outputs its raw data in frames which contain a raw point cloud, a modular processing pipeline was developed. This pipeline consists of different modules implementing the needed stages for pre-processing the sensor's data, filtering the point cloud, clustering and object detection. The modular design allows the modification and adaption to different environments.

In a pre-processing stage, the sensor's raw data frames, obtained via UART, are decoded and translated into frames containing individual points in a usable format with information about their x, y, z, v_r and SNR values. They are then passed into a "Frame Aggregator" which stores a certain amount of frames in order to supply the later stages with the data from multiple frames and thus compensate for possible data sparsity. The aggregated data is filtered by multiple filtering stages, starting with static parameters for the spatial dimension and the SNR, followed by a dynamic filtering stage using a comparison of the vehicle's estimated self-speed to the points' speeds for filtering. The filtered points are then clustered using a two-stage approach and forwarded to the brake controller for object detection. The reason for the 2 stages of clustering is that for any given object that may be detected there would be noise or segmentation caused by some space between objects. So this 2 stages help us discard information that could be just a reflection and focus into the points that are considered as objects. This being done by having a permissive first stage, and afterwards a strict clustering stage to be able to label an object with a cluster ID for further processing.

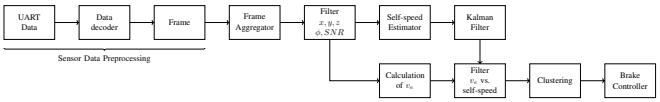


Fig. 13: Block diagram of the pipeline

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.

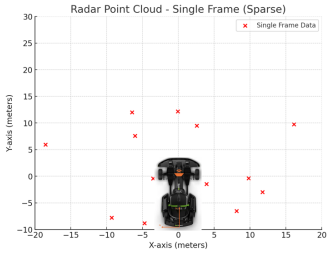


Fig. 14: Single Frame visualization.

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.

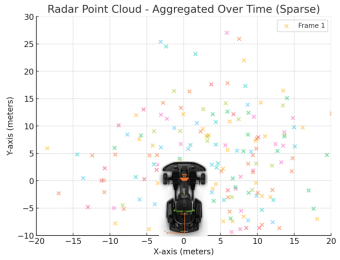


Fig. 15: Frame Aggregation visualization.

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.

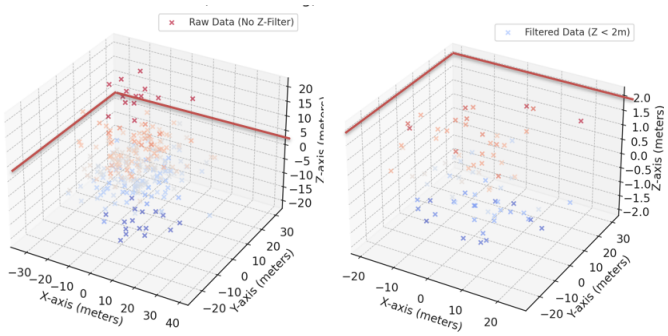
B. "x, y, z, ϕ, SNR" Filter

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

C. 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:

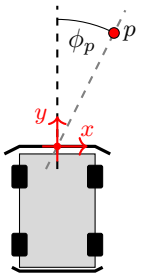


Fig. 17: Definition of the angle ϕ_p

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.

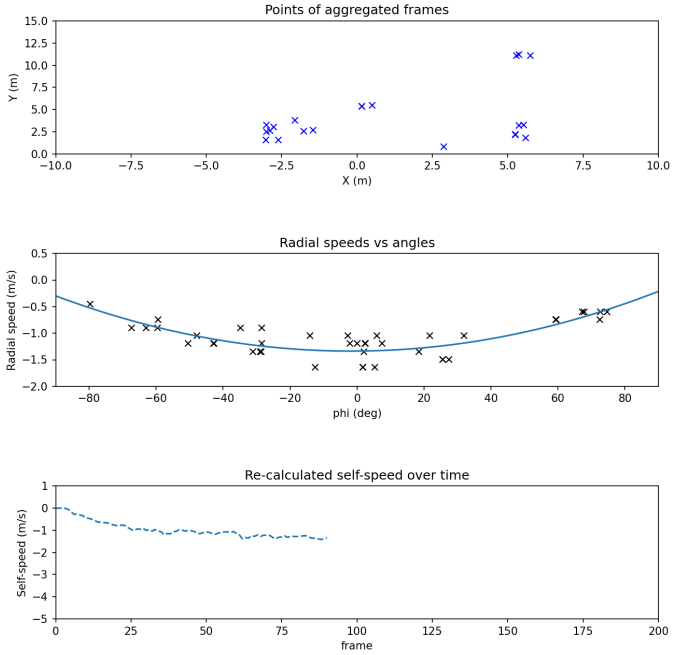


Fig. 18: Test run with recorded data and visualization.

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.

D. 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.

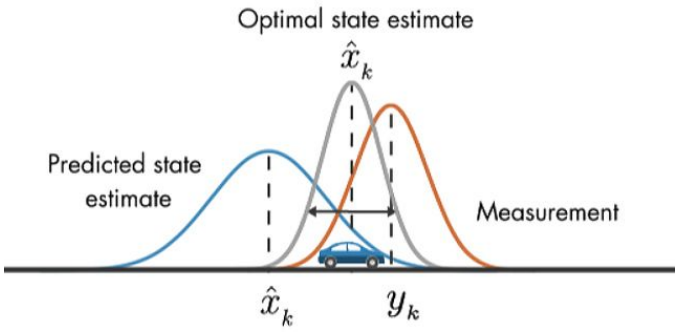


Fig. 19: Kalman filter prediction of the state using estimations. From: [15].

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.

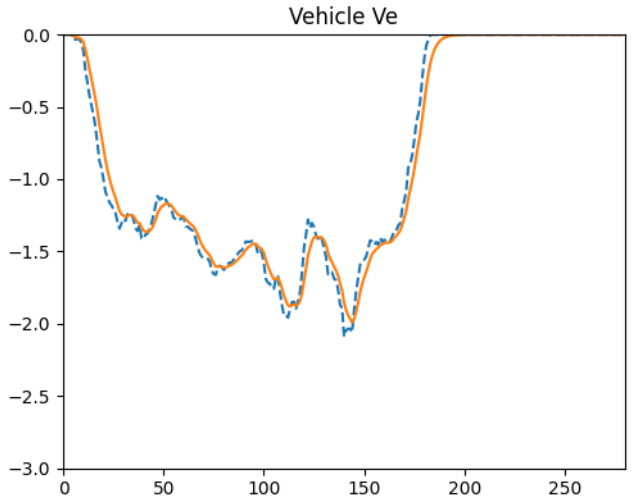


Fig. 20: Raw self-speed vs. self-speed after Kalman filter.

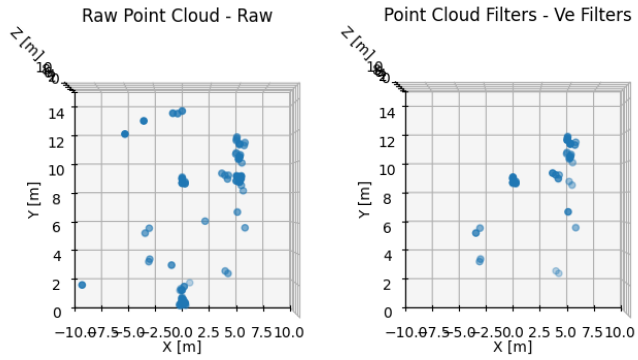
E. Calculation of v_e and v_e vs. Self-Speed Filter

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 17). 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.



(a) Input (b) Output

Fig. 21: Example visualization of the input and output data before and after passing the v_e vs. self-speed filtering stage.

F. 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.

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.

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 II: Comparison of Clustering Algorithms

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:

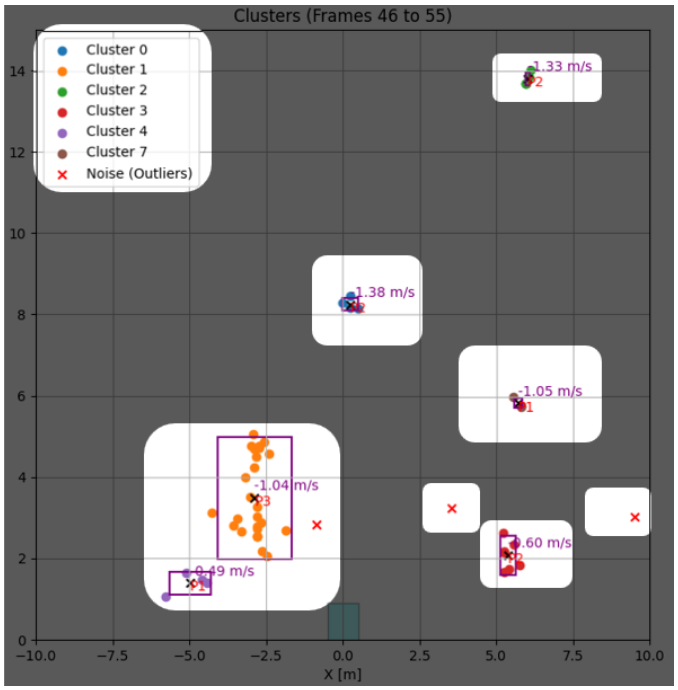


Fig. 22: First clustering stage

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.

G. 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

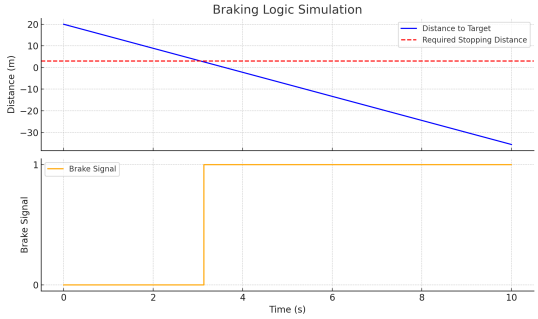


Fig. 23: Braking decision based on stopping distance estimation.

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.

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).