Parameter Tuning and Customization Guide for the xWRLx432 Motion/Presence Detection Demo

Version 3.0
April 2024

Texas Instruments, Incorporated
12500 TI Boulevard
Dallas, Texas 75243 USA

Version History

Version	Date	Comment
1.0	December 2022	Initial version.
2.0	April 2023	Target tracking and classification modules are added.
3.0	May 2024	Updates for ES2 silicon

Table of Contents

1 Introduction and Scope

This document provides a high-level overview of the signal processing chain of the motion/presence detection demo and presents a guide for the user to tune the performance of this demo for different use cases and environments. It aims to explain the configurable processing chain parameters, which the user can tune to improve the overall demo performance. It is important to note that the scope of this document is limited to the performance tuning level. The implementation details of the processing chain, including the execution flow, memory management, etc., are given in a separate demo implementation guide. Please refer to the mmWave software development kit (SDK) [1] for the implementation details of the demo.

Section 2 briefly introduces the overall signal processing chain. In Section 3, we detail the parameters that can be configured through the command-line interface (CLI) of the device. Section 4 describes some commonly encountered cases in which the user may need to tune the parameters to obtain optimal performance in the motion/presence detection demo. As discussed in the following sections, although the name is preserved as motion/presence detection in the updated release, significant number of new features are added into the demo to extend the functionality beyond basic motion/presence detection, including target tracking, motion classification, and more.

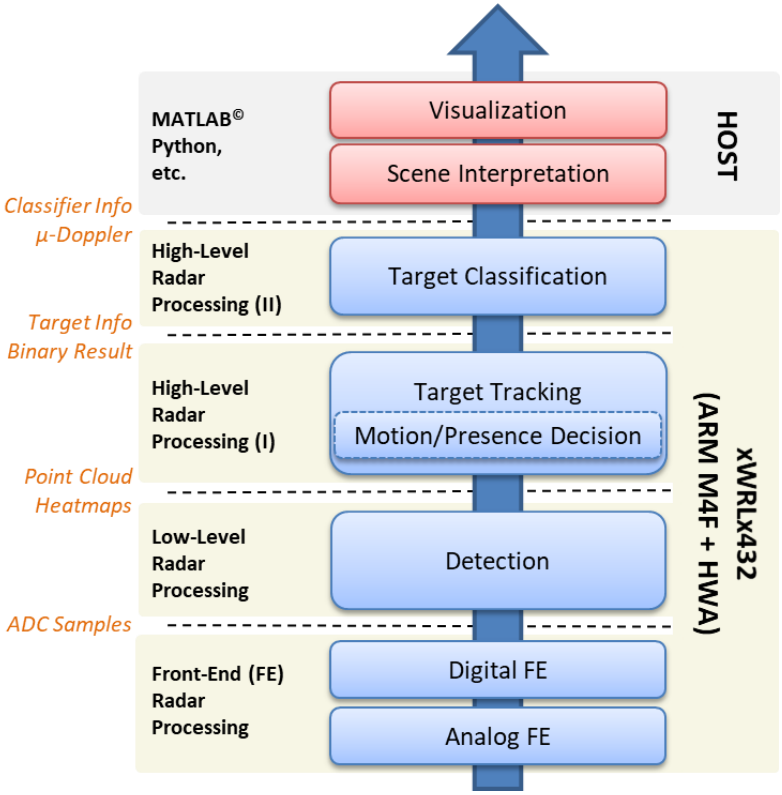


Figure 1. The signal processing layers of the motion/presence detection demo.

In the motion/presence detection demo running on the Texas Instruments' xWRLx432 mmWave radar sensors, different processing layers are offered based on the use case requirements, as illustrated in the flow diagram in Figure 1.

At the lower level, the front-end radar processing layer handles the radar signaling parameters such as chirp, frame, modulation, etc. The detection layer is responsible for sensing multiple reflections from the three-dimensional (3D) environment around the sensor and delivering a set of measurement vectors, known as the point cloud, representing the real-life targets in the scene. The analog-to-digital converter (ADC) data (i.e., the beat signal) from the sensor front-end is the input of the detection layer. Each measurement vector generated by the detection layer represents a reflection point with range, azimuth, elevation, radial velocity (i.e., Doppler), and signal-to-noise ratio (SNR). A high-level motion/presence decision layer then processes the point cloud to run the logic that decides whether a user configured region of interest (ROI) is occupied or not.

We develop this demo to run on the xWRLx432 mmWave radar sensors. As depicted in Figure 1, the processing chain on the device provides the point cloud data and other detection layer data (heatmaps, etc.) to the host. The functional blocks that provide the final motion/presence decision using the generated point cloud are also running on the device in real-time. In some use cases, a tracking layer [2] implemented on the device can process the point cloud to localize and track the detected targets in the scene. Similarly, it may be important to discriminate between humans moving through a scene and other movements generated by non-human motion sources. In this release, a classification layer is also developed to run on the device in real-time to perform such object classification tasks. In the current version of the demo, only the data visualization to the user is implemented on the host in a graphical user interface (GUI) based application. As illustrated in Figure 1, all other processing blocks are running on the device in real-time.

2 Signal Processing Algorithm

This chapter will summarize the signal processing algorithm of the motion/presence detection demo. It will be divided into the following parts:

a) Overall signal processing flow: a high-level description of the overall processing chain.
b) System geometry and output data: the coordinate systems used and the output data provided.
c) Chirp processing: functional details of each sub-blocks in the chirp processing step.
d) Range-gate processing: functional details of each sub-blocks per range-bin processing.
e) Peak detection processing: functional details of each sub-blocks in the peak detection step.
f) High-level processing: functional details of each sub-blocks in the high-level processing step:
    i) Motion/presence detection: to give a binary motion/presence decision in the given ROI.
    ii) Target tracking: to localize and track multiple targets in the configured scene.
    iii) Target classification: to classify the tracked objects as human or non-human.

2.1 Overall Signal Processing Flow

Figure 2 illustrates the typical processing blocks and data flow for the motion/presence detection demo. Subsequent sections will describe the details of the processing flow. As depicted in Figure 2, the chirp processing module running on the Hardware Accelerator (HWA) takes the ADC samples per chirp from the front end for all the physical receive channels (antennas), performs de-multiplexing the BPM-MIMO scheme (if enabled) and range fast Fourier transform (FFT), then stores the output to the radar cube. Range gate processing blocks are then called per range gate/bin. For each range gate, phase/gain calibration for all the antennas is applied. Doppler and angle processing over the entire chirps and antennas within the frame are performed by the HWA configured by the M4F core, depending on the use case parameters. It is important to note that different antenna patterns may be used for different use cases. Hence, the user needs to reconfigure (will be discussed in Section 3) the angle processing blocks to accommodate different antenna patterns. It is important to note that Figure 2 illustrates the processing flow at a high level. Due to the implementation aspects, the order of the blocks can be changed in the entire flow.

After the range gate processing, the detection matrix (i.e., heatmap) is created in the spatial range-angle domain. The HWA performs the peak detection processing on the generated range-azimuth heatmap with some operations assisted by the M4F core to create the output peak list and other attributes of the detected peaks. The final peak detection list is then shipped to the host to feed the higher processing layers. The current peak detection processing is performed in the range-azimuth domain, assuming a typical angular accuracy created by 2 TX and 3 RX channels will be dominant in the azimuth domain. For the use cases that need motion localization in 3D space, the elevation estimation is done only for the detected points in the range-azimuth domain.

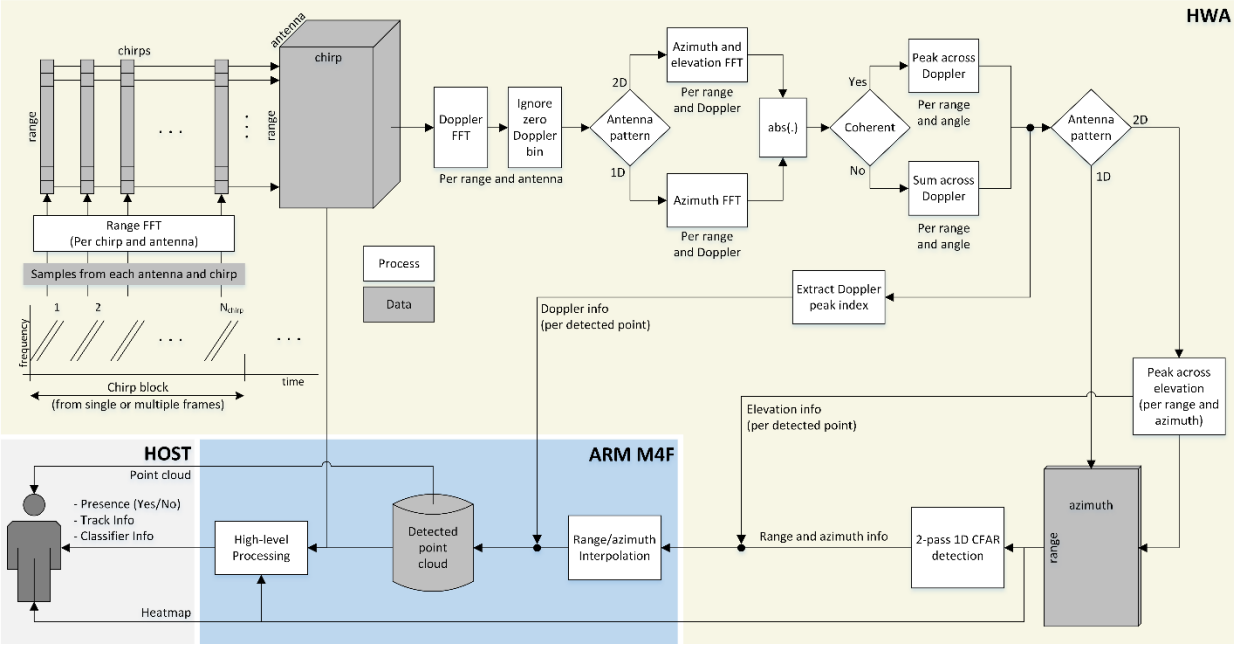


Figure 2: Typical radar processing chain on xWRLx432 to create the point cloud and the higher level of data (motion/presence decision, tracking, and/or classification information).

As illustrated in Figure 2, the motion/presence detection demo utilizes the 3D radar data cube as input, similar to the conventional radar signal processing chains. The first axis of the radar data cube corresponds to the range spectrum computed from time-domain ADC samples of each chirp signal. It is important to emphasize that the zero-padded range FFT size is always the nearest next power of two of the number of ADC samples. The second axis of the radar cube holds the spatially sampled signals using multiple virtual antennas to estimate the direction-of-arrival (DOA) of the targets. The third axis (typically known as the slow-time domain) stores a set of chirps from each virtual channel to provide a basis for estimating the motion (i.e., Doppler speed) of the targets.

The main idea of the signal processing chain is to detect both major (walking, etc.), minor (typing, etc.), and very fine (sitting still, sleeping, etc.) motions in the same scene. Increasing the velocity resolution can help improve the motion sensitivity but also needs a more chirping window in a single frame, resulting in more power consumption. In order to achieve a longer chirping window with less power when detecting minor motions, the detection layer chain running on HWA and M4F discussed in Figure 2 will run utilizing the combined subsampled chirp blocks from the current and previous frames to increase the chirping window for minor and very fine motions.

The motion/presence detection demo (both for binary motion/presence decision and target tracking) will support the following three modes depending on the use case requirements:

- Auto mode: In this mode, the detection layer chain will run two times using both the radar cube from the current frame with an optimized chirping window for major motions and the radar cube created across the current and previous frames to detect the minor motions in the scene. The

generated point cloud will be merged to be used in the proper state machines running in the high-level processing blocks.

- Major-only mode: In this mode, the detection layer chain will run one time using the radar cube from the current frame with an optimized chirping window for major motions. This mode will generate a single point cloud set to be utilized by the high-level processing blocks.
- Minor-only mode: In this mode, the detection layer chain will run one time using the radar cube created across the current and previous frames. This mode will generate a single point cloud set to be utilized by the high-level processing blocks.

Figure 3 illustrates the generation process of the 3D radar data cubes for both major and minor motions. In this example, four chirps per TX antenna are configured within a single frame to create the radar cube for major motion detection. To handle the minor motions, two frames are combined (two chirps per TX and per frame) to create the radar cube for minor motion detection. As discussed in Section 3, the number of chirps at each frame or the number of frames combined in minor mode are all configurable depending on the use case requirements.

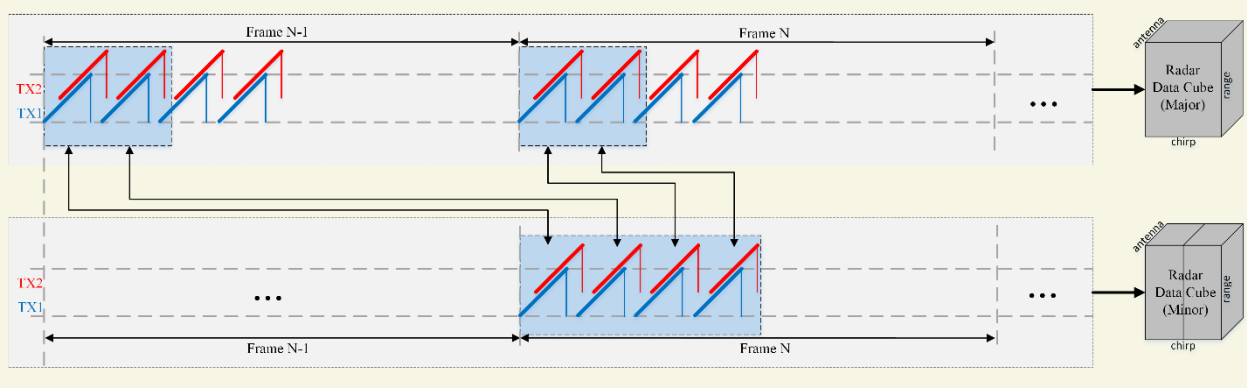


Figure 3: Radar cubes generation flow from the current frame and across multiple frames to detect major and minor motions, respectively.

The detected point clouds of each processing are then merged to be processed in the high-level processing logics, as illustrated in Figure 4.

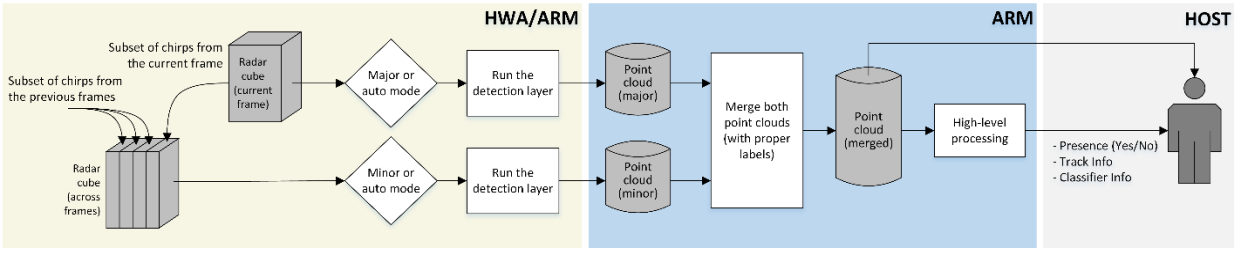


Figure 4: Using two radar cubes for higher-level processing.

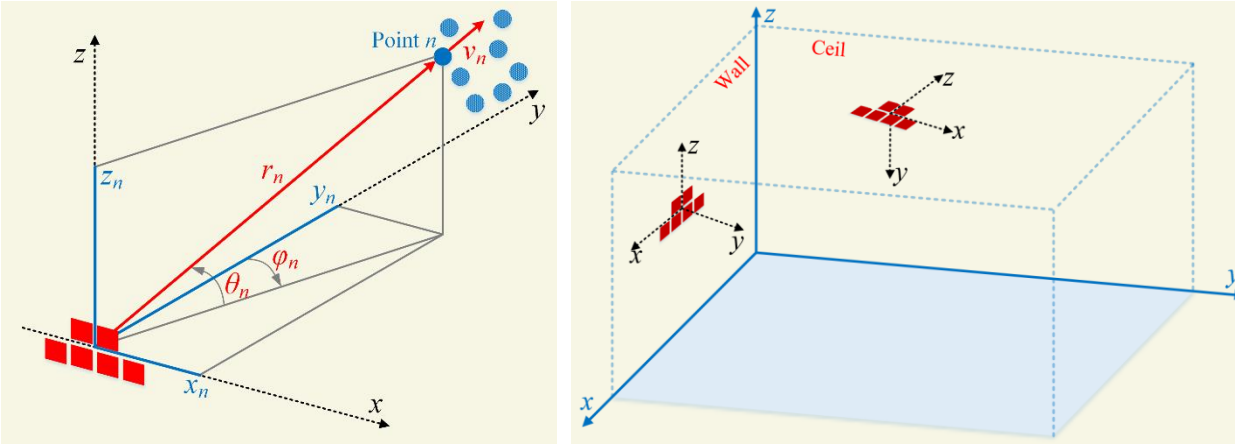
2.2 System Geometry and Output Data

The motion/presence detection demo running on the xWRLx432 mmWave sensors supports the antenna pattern on the xWRLx432 evaluation module (EVM) [10], which can also be extended to different antenna patterns in a flexible way, as discussed in the rest of the document. Utilizing the two-dimensional (2D) geometry of this antenna pattern, the detection layer outputs the 3D point cloud in spherical coordinates referenced to the sensor positioned at the established 3D Cartesian coordinate system's origin. Figure 5a illustrates the system geometry of a single reflection point *n* for the mentioned antenna pattern. Each generated point from the target is represented by a range, angle (both azimuth and elevation), radial velocity, and SNR:

- Range r, $r_{min} < r < r_{max}$
- Azimuth angle φ, $-\emptyset_{max} < \varphi < +\emptyset_{max}$
- Elevation angle θ, $-\theta_{max} < \theta < +\theta_{max}$
- Radial velocity v, $-v_{max} < v < v_{max}$
- SNR

The azimuth (φ) is defined as the angle from the y-axis to the orthogonal projection of the position vector onto the xy-plane. The angle is positive, going from the positive y-axis toward the positive x-axis. The elevation (θ) is defined as the angle from the projection onto the xy-plane to the vector. The angle is positive, going from the xy-plane to the positive z-axis. It is important to note that the coordinate systems in Figure 5a are anchored to the radar sensors. In other words, the sensor's location contains the origin of its local coordinate system. The detected points in the established sensor coordinate system follow these axes conventions:

- The positive x-axis points to the right, as viewed when facing forward,
- The positive y-axis points forward from the sensor (i.e., boresight),
- The positive z-axis points up from the xy-plane to maintain the right-handed coordinate system.



(a) (b)

Figure 5. (a) The system geometry of the point cloud in the 3D coordinate system. (b) The global and local coordinate system.

The motion/presence demo supports various sensor mount configurations in a real-world scene depending on the use case. To interpret the detected point cloud, the motion/presence decisions, or the tracked target information in the global (i.e., world) coordinate system, the user needs to understand how the radar is mounted in the global frame. The sensor mounting frame in Figure 5a will have some offset by a displacement from the global frame origin. Moreover, the sensor mounting frame may also have an angle rotation (i.e., azimuth or elevation tilt) in the global frame. The generated information (point cloud, tracker states, etc.) will be streamed in local coordinates to be handled by the host. On the other hand, the high-level processing layers running on the device also handle the conversion step from local coordinates to global coordinates to give a binary motion/presence decision within the configured ROI in world coordinates.

Figure 5b illustrates the relationship of local and global coordinate systems in two different scenarios (wall and ceil mount). The thick blue solid lines represent the global frame's coordinate axes, while the sensors carry their local coordinate system depicted with dashed black lines. The user-configurable displacement and rotation parameters are discussed in the configuration parameters tuning section (Section 3.4 and Section 3.5).

SNR in the generated point cloud is defined as the ratio of reflected signal power to the noise power in the linear scale. A ratio higher than 1:1 (i.e., greater than 0 decibels (dB)) indicates more signal than noise. In the radar domain, SNR is usually a function of the target's radar cross-section (RCS), and a larger SNR from a highly reflected object results in higher measurement accuracy. Please refer to the mmWave training series [3] for the details about SNR.

As discussed in Section 1, point cloud is not the only data generated by the motion/presence detection demo. The processing chain implemented on the device in real-time also has the option of streaming tracked target data (if the tracker layer discussed in Section 2.7 is enabled) or the classifier layer data including the µ-Doppler information of each tracked object and the track labels (human or non-human) (if the classifier layer discussed in Section 2.8 is enabled). Figure 6 depicts an example set of outputs available in the motion/presence detection demo. For the complete list of data provided by the motion/presence detection demo, please refer to the demo implementation guide in mmWave SDK [1].

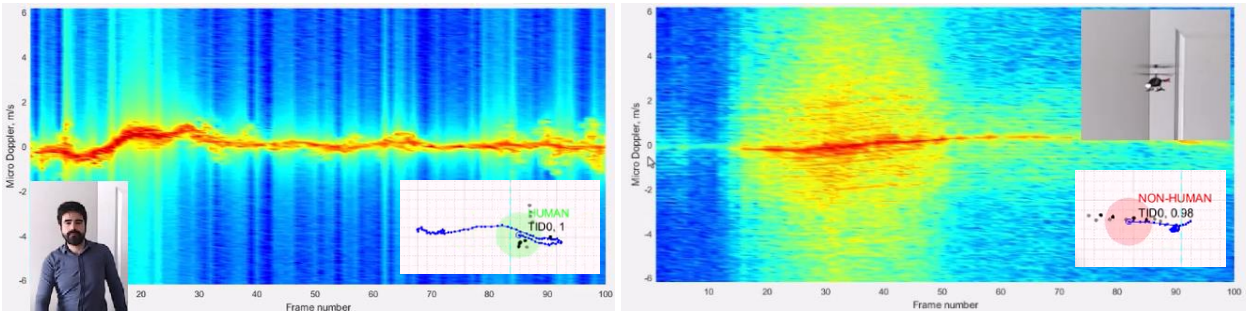


Figure 6: The high-level data examples including tracking (with target trajectories), µ-Doppler spectrograms and the classifier tags (human vs non-human) of an example scenario.

2.3 Chirp Processing

The chirp processing step is applied per chirp on the ADC samples from all TX-RX pairs (i.e., virtual RX channels/antennas) to create the range spectrum of the target scene. The processing block diagram is shown in Figure 7.

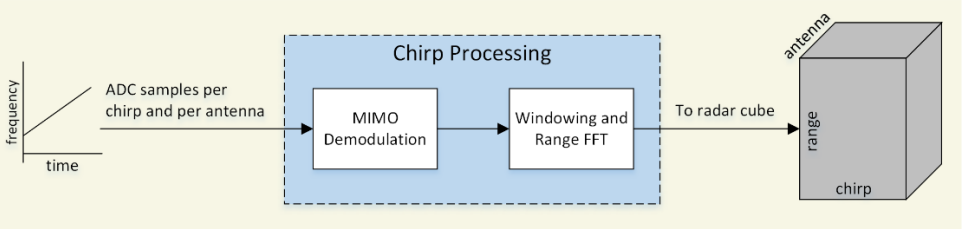


Figure 7: Chirp processing flow.

Chirp processing is called for all chirps within the frame and for all virtual antenna channels. Data inputs are the ADC samples from the front-end. The data output is the range domain signal that is stored in the radar cube.

2.3.1 MIMO Demodulation

To separate the signals corresponding to different TX antennas in the 2 TX and 3 RX MIMO schemes, both BPM-MIMO and TDM-MIMO will be supported in the xWRLx432 processing chain. Figure 7 depicts both TDM-MIMO and BPM-MIMO schemes with two TX antennas. The following sections briefly summarize both modulation schemes. For details about the timing parameters in Figure 8, refer to the SDK [1] and the device firmware package (DFP) interface control document (ICD) [6].

Figure 8 also illustrates the typical timing diagram for two consecutive chirp loops. It is important to note that two chirps per MIMO scheme (both BPM and TDM) are close to each other in the time domain (i.e., a short idle time 2) to reduce the angle estimation errors caused by the phase difference across transmit antennas. On the other hand, a longer idle time between two consecutive chirp pairs is used (i.e., idle time 1) to improve the velocity resolution and detect minor scene motions.

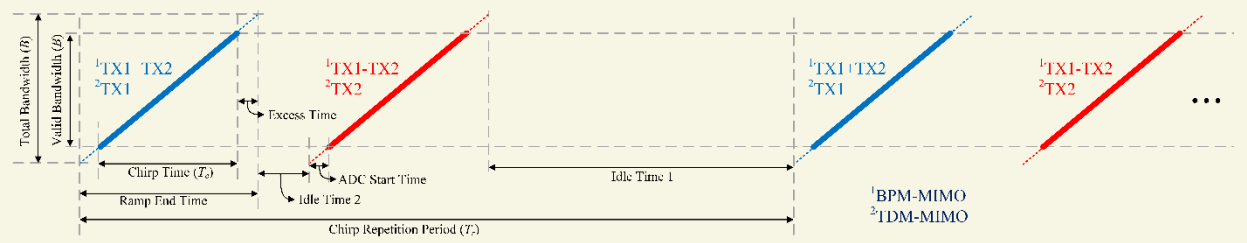


Figure 8: BPM-MIMO and TDM-MIMO schemes with the detailed timing diagram for two consecutive chirp loops.

2.3.1.1 TDM-MIMO

In the TDM-MIMO scheme, only one TX antenna is active in a given chirp, and all the TX antennas are enabled in the frame in an alternating manner. For example, for 2 TX (TX1 and TX2), chirp 0 has only TX1 enabled, chirp 1 has only TX2 enabled, chirp 2 has only TX1 enabled, and so on. Therefore, no additional demodulation step is needed in the TDM-MIMO scheme because the chirps from the TX antennas are separated in the time domain and stored in the radar cube properly.

2.3.1.2 BPM-MIMO

Similar to TDM-MIMO, in the BPM-MIMO scheme, a frame consists of multiple blocks, each consisting of 2 chirp intervals. However, unlike in TDM-MIMO, where only one TX antenna is active per chirp interval, two transmit antennas are active in each chirp interval. For the BPM-MIMO depicted in Figure 8, in the even time slots (0, 2, ...), both transmit antennas are configured to transmit with positive phase, i.e. (TX1, TX2) = (+, +). In the odd time slots (1, 3, ...), the transmit antennas are configured to transmit with phase (TX1, TX2) = (+, -).

Therefore, when the BPM mode is enabled, the signal processing chain must decode the received chirp pairs to separate the transmission from each TX antenna. As shown in Figure 7, the MIMO demodulation block reads the data for a range gate and performs the MIMO demodulation, as summarized below. Output is the per range gate MIMO demodulated chirp domain signal for all virtual RX channels.

Let TX1 and TX2 represent chirp signals from two TX antennas. In time slot zero, a combined signal S_A= TX1+TX2 is transmitted. Similarly, a combined signal S_B= TX1-TX2 is transmitted in time slot one. Using the corresponding received signals (S_A and S_B) at a specific received RX antenna, the components from the individual transmitters are separated using

- TX1=(SA+SB)/2, and
- TX2=(S_A-S_B)/2.

With simultaneous transmission on both TX antennas, the total transmitted power per chirp interval is increased, and it can be shown that this translates to an SNR improvement of 3dB. On the other hand, the average power consumption also increases with the BPM-MIMO mode compared to the TDM-MIMO scheme.

2.3.2 Windowing and Range FFT

As shown in Figure 9, the windowing and range FFT block takes the time-domain ADC data and performs windowing and range FFT for all the virtual RX channels. The output will be stored in the radar cube. In xWRLx432, real-only samples are available in the time domain. Therefore, the fixed real windowing coefficients will be configured to multiply with the real samples within the range processing (per chirp or RX channel). After the FFT processing, only half of the range spectrum will be stored in the radar cube (due to the symmetry). The range spectrum output of the range FFT sub-block has complex samples.

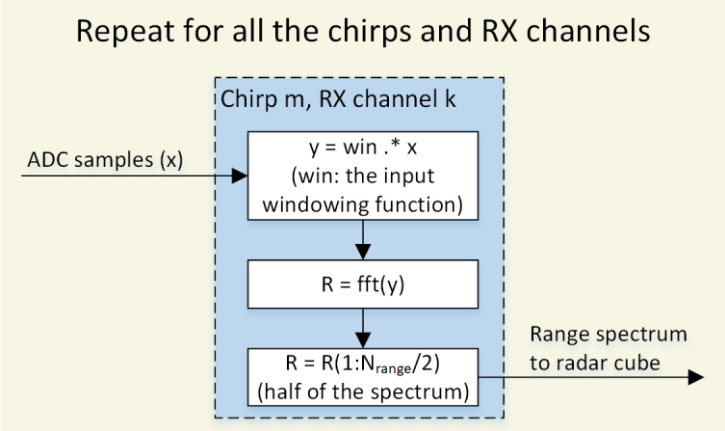


Figure 9: Functional flow for windowing and range FFT.

2.4 Range Gate Processing

Range gate processing is called per range bin/gate after chirp processing. Data input is the per range gate samples from the radar cube for all the chirps and RX channels. Output is the power spectrum (i.e., heatmap) in the azimuth domain with the elevation and Doppler bins where this spectrum resides. This section is further divided into subsections to provide functional descriptions for MIMO demodulation, RX gain/phase compensation, Doppler processing, and angle processing. The functional flow for range gate processing is shown in Figure 10.

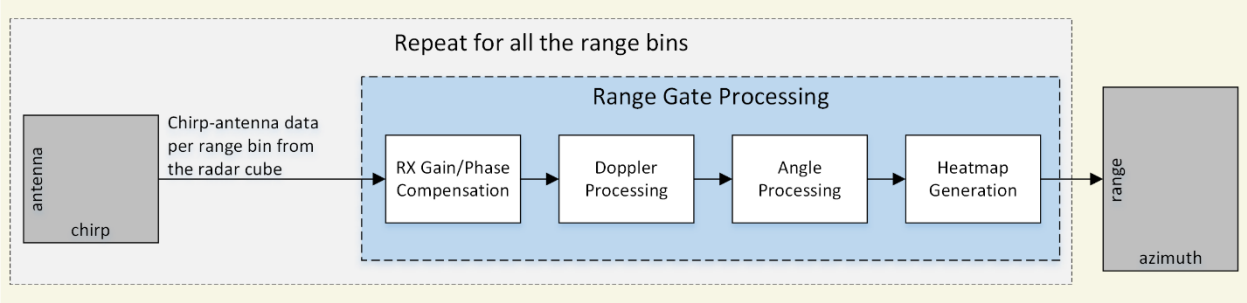


Figure 10: Functional flow for range gate processing.

2.4.1 RX Gain/Phase Compensation

As shown in Figure 11, RX gain/phase compensation is done per chirp for all the RX channels by multiplying the complex vector coefficients with each antenna sample. The length of the calibration vector should be the $N_{TX} \times N_{RX}$, where N_{TX} is the number of physical TX antennas (two by default), and N_{RX} is the number of physical RX channels (three by default). The calibration coefficient vector is assumed to not change within the range gate. Section 3.2.6 describes how to estimate and apply these phase compensation factors to the processing chain.

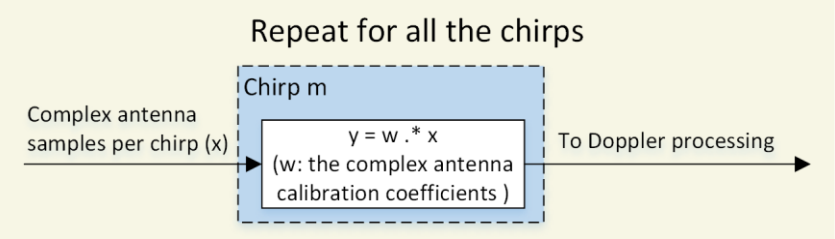


Figure 11: Functional description for RX Gain/Phase Calibration per range gate.

2.4.2 Doppler Processing

As shown in Figure 12, the Doppler processing block reads the MIMO demodulated and RX channel calibrated data from RX gain/phase compensation block for a range gate, performs windowing and Doppler FFT over the consecutive chirps in the radar data cube per RX antenna, and zeroized the first Doppler bin corresponding to the static clutter. The output sent for angle processing is the static clutter removed Doppler domain signal for all virtual RX channels per range gate. In the Doppler domain, a fixed rectangular window is used by default. The static clutter removal step by zeroing the first Doppler bin is applied to remove the purely static objects, such as chairs and tables, from the scene and leave only the signals backscattered from the moving objects.

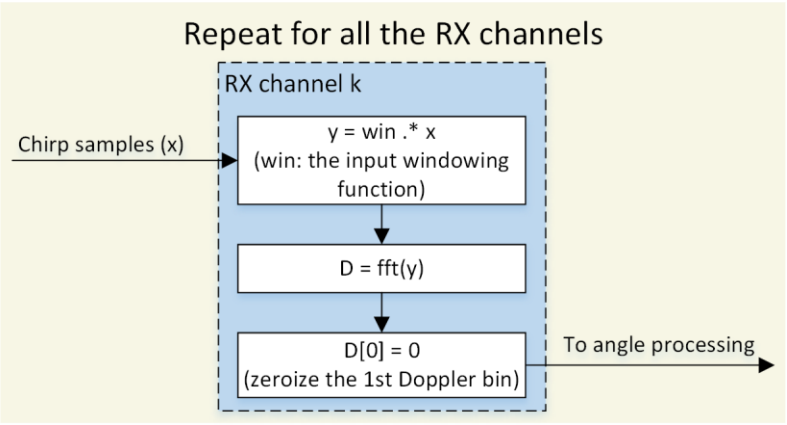


Figure 12: Functional flow for Doppler processing per range gate.

It is important to note that the clutter removal approach implemented by zeroing the first Doppler bin can only work if the total number of chirp samples in the Doppler processing block is a power of 2, the Doppler FFT size is the same as the number of chirps, and no windowing is used before the Doppler FFT. In this case, there will be no zero paddings in the radix-2 Doppler FFT processing on the HWA. Hence, the zero Doppler bin will correspond to the average across chirps, which can be used as a static clutter estimation.

2.4.3 Angle Processing

The angle processing block takes the $N_{Doppler}$ by N_{TX}×N_{RX} matrix per range bin from the Doppler processing block. As shown in Figure 13, the N_{TX}×N_{RX} antenna samples per Doppler bin are partitioned into multiple azimuth rows depending on the antenna pattern. A zero-insertion block takes the antenna samples and inserts zeros for possible missing samples for each azimuth row.

As shown in the azimuth FFT block in Figure 13, the input vector from the zero-insertion block will also be padded with zeros depending on the azimuth FFT size. The azimuth FFT is then performed to the per Doppler bin and per azimuth row input. No windowing function is applied before the azimuth FFT.

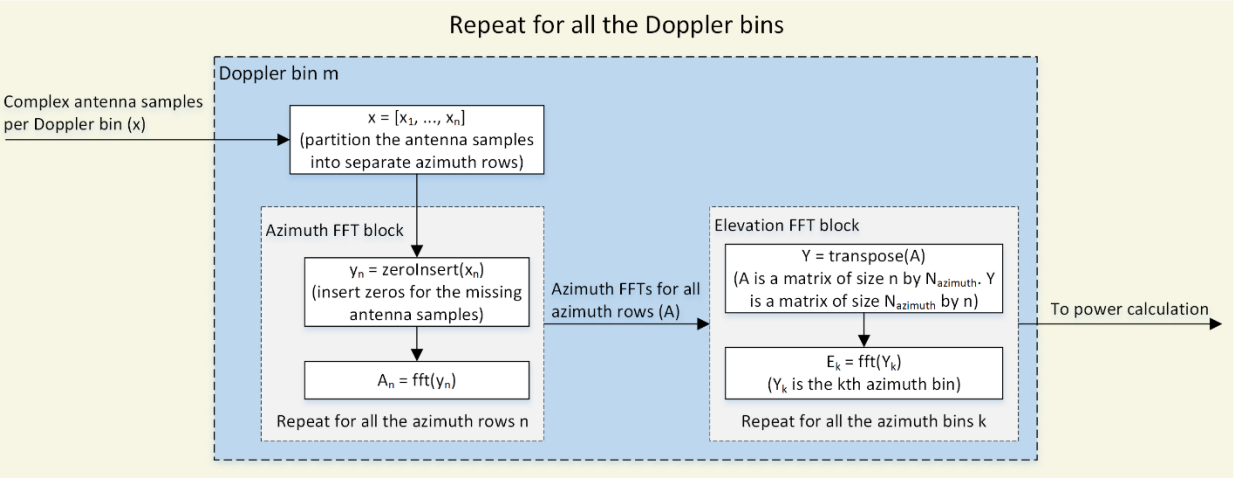


Figure 13: Functional flow for angle processing per range gate.

Depending on the antenna pattern and the use case (i.e., if a 2D antenna pattern is used and the use case requires an estimation in 3D space), the elevation processing will be handled by the elevation FFT block in Figure 13. The input vectors from the azimuth FFT block will be padded with zeros depending on the elevation FFT size. The elevation FFT is then performed to the per azimuth bin input. No windowing function is applied before the elevation FFT. If a one-dimensional (1D) antenna pattern (along the azimuth domain) is used, the elevation FFT block can be skipped. Similarly, if a 2D pattern is used, but an estimation in 3D space is not needed, the elevation FFT block can be skipped, and only a single azimuth row with the greatest number of antennas in the azimuth direction can be used to create a single azimuth spectrum per Doppler bin. Section 3 describes how to configure the desired antenna patterns (or subset of the available antennas) and the azimuth/elevation FFT sizes.

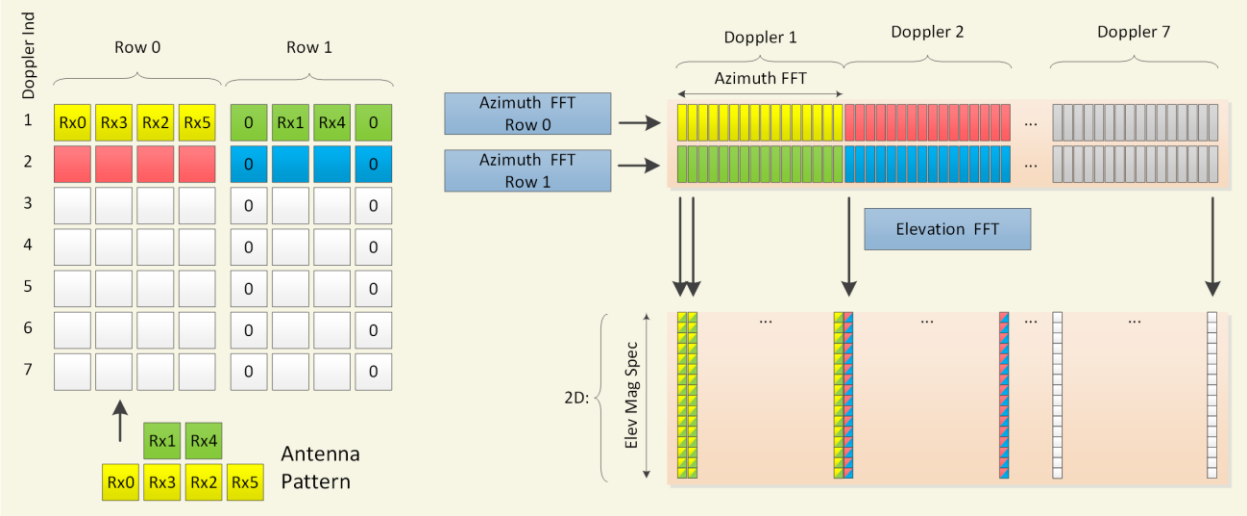


Figure 14: The detailed angle processing scheme for an example 2D antenna pattern.

Depending on the MIPS (i.e., that also affects the power consumption) and memory requirements for each use case, a small azimuth and elevation FFT size may be required (the MIPS and memory requirements will be discussed in the processing time memory requirements section). This will create coarse angle steps in the azimuth-elevation grid. The azimuth interpolation block (mentioned in the range-azimuth interpolation section) will improve the azimuth estimation accuracy.

2.4.4 Heatmap Generation

The heatmap generation block takes the complex $N_{Doppler} \times N_{azimuth} \times N_{elevation}$ matrix per range bin from the angle processing block and creates the real azimuth domain power spectrum per range gate to be used in the detection processing.

2.4.4.1 Power Calculation and Doppler Projection

The power calculation block first calculates the power spectrum of the Doppler, azimuth, and elevation FFTs output by taking the magnitude of each complex sample. Figure 15 shows the functionality of the projection of the 3D Doppler-azimuth-elevation power spectrum from the power calculation block into the 2D azimuth-elevation spectrum per range gate. Depending on the coherent gain needed in the Doppler domain for different use cases, the Doppler domain power spectrum is projected into a single scalar per angle bin (azimuth-elevation) either by a non-coherent summation (by default) or a coherent peak selection step. The power calculation and Doppler projection block output is the azimuth-elevation domain power spectrum (or the azimuth domain power spectrum if the elevation estimation is skipped). Depending on the Doppler projection method, the Doppler index is also provided by this block. The peak index is provided as the Doppler estimation if the maximum peak selection is configured. If the mean across Doppler is configured, the Doppler estimation is set to zero by default.

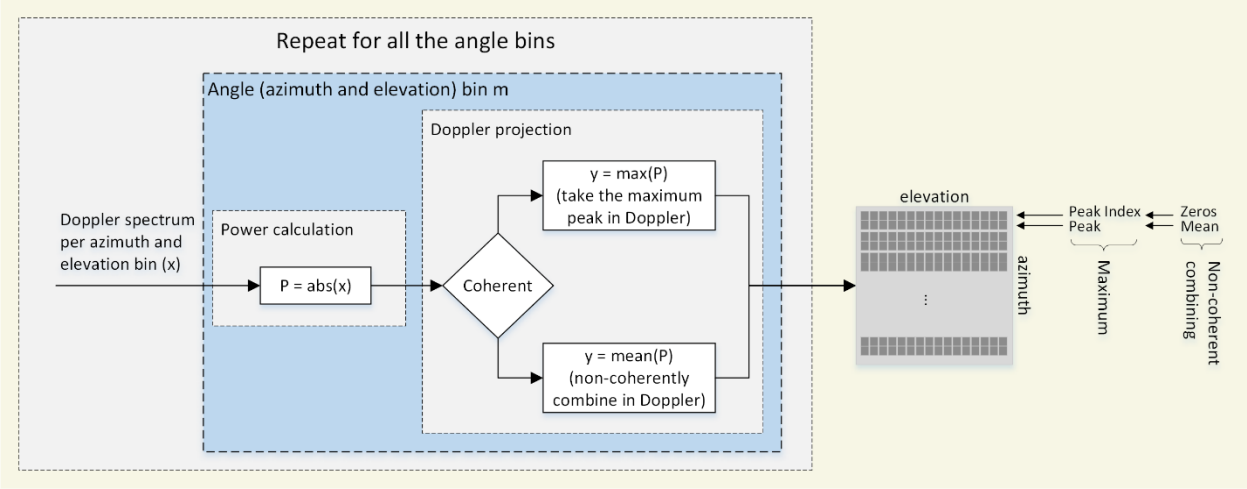


Figure 15: Functional description of power calculation and Doppler projection.

2.4.4.2 Elevation Estimation

The azimuth-elevation domain power spectrum is calculated from the Doppler projection block per range gate. The elevation estimation block then performs a maximum peak search in the elevation domain per azimuth angle bin. The output will be an array of peak descriptors containing the strongest (i.e., maximum) peak in the elevation spectrum and the corresponding peak indices per azimuth angle, as illustrated in Figure 16.

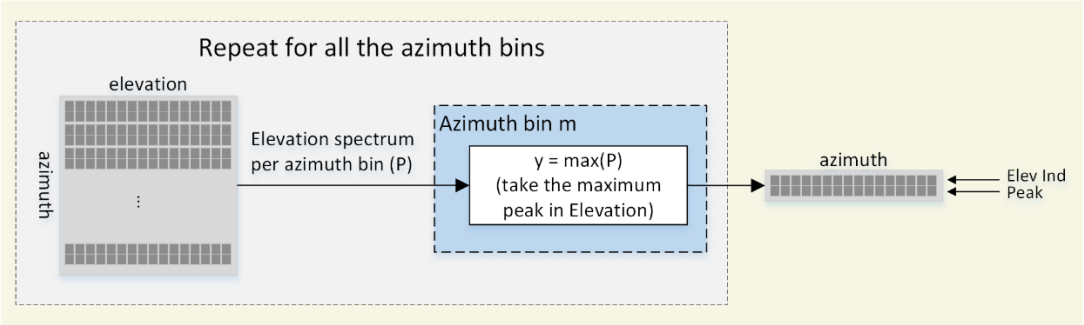


Figure 16: Functional description of elevation estimation.

2.5 Peak Detection Processing

This step applies a 2-pass constant false alarm rate (CFAR) algorithm to the 2D range-azimuth heatmap for detection. In this approach, the detected points created by the first-pass CFAR in the range domain are confirmed by a second-pass search step in the azimuth domain. The second-pass consists of local peak search and sidelobe check steps based on configurable logic to confirm the detected points created by the first-pass range-CFARs. The peak detection processing block also adopts an interpolation step on the detected range and azimuth points to improve the accuracy of the estimation without increasing the FFT size.

The functional block diagram of the peak detection processing is shown in Figure 17. Data input of the peak detection processing is the range-azimuth domain power spectrum generated by range gate processing blocks. Output is an array of local peak descriptors that contain range and azimuth information of each detected peak.

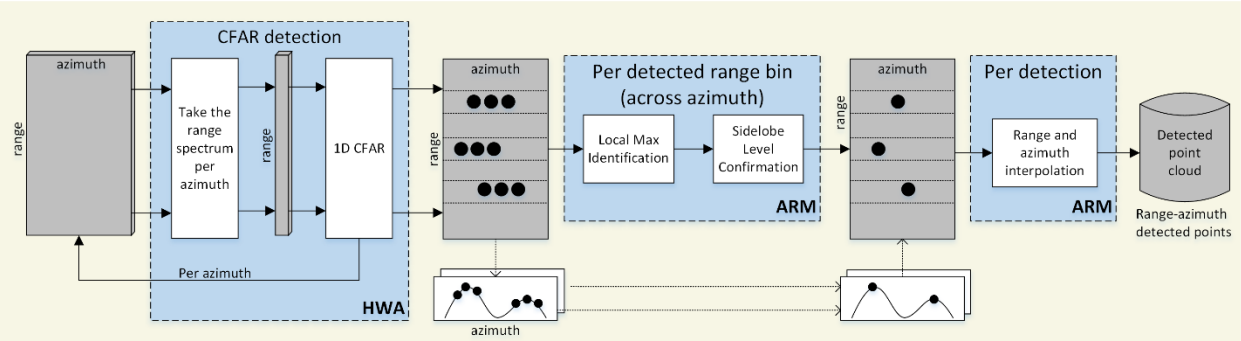


Figure 17: Functional description of peak detection processing.

Combining the Doppler and elevation index generated in the range gate processing block, a full set of the point cloud is generated to be used in the high-level processing chain.

2.5.1 CFAR Detection

Figure 17 shows the functionality of the CFAR detection processing block. The following information will be gathered in a peak descriptor structure for all the detected points:

- Range index (rngIdx) and azimuth index (azIdx) of all the peaks detected by the first-pass range-CFARs per azimuth bin.
- 2D neighborhood power spectrum around each detected point, which is a real value array containing the power for the 2D region of [(rngIdx-1):(rngIdx+1), (azIdx-1):(azIdx+1)]. The neighborhood should be copied in edge-wrapped-around fashion in both dimensions.
- SNR estimate of the detected peaks is computed by dividing the peak value of the search window by the noise estimation.

2.5.2 Local Max Identification

The local max identification block takes the range-azimuth domain power spectrum as input and performs a local maximum search along the azimuth domain per detected range bin in the first-pass CFAR processing. The local max check is done in an edge-wrapped-around fashion along the azimuth domain power spectrum. The following condition has to be satisfied for an element $P_{i,j}$ in input range-azimuth heatmap P to be declared as a local max:

- $P_{i,j} \geq P_{i,j+1}$ AND $P_{i,j} \geq P_{i,j-1}$, in edge wrapper around fashion

The filtered range and azimuth index of the detected peaks are the output of the local max identification block. In the current functional split, the local max identification block will be running on M4F.

2.5.3 Sidelobe Level Confirmation

This processing block performs an absolute max search in the azimuth domain power spectrum per detected range bin. The absolute max is then multiplied by a configurable gain (0 to 1) to compute a threshold used in the sidelobe level confirmation block. $Thr_{A,i}$ will be the output from this block for the i^{th} range bin. Local maximums are then filtered based on these thresholds. The following condition has to be satisfied for an element $P_{i,j}$ in input range-azimuth heatmap P to be declared as not a sidelobe of the absolute maximum peak.

- $P_{i,j} \geq Thr_{A,i}$

In the current functional split, the max search is performed on HWA, while threshold calculation and sidelobe confirmation runs on M4F.

2.5.4 Range-Azimuth Interpolation

In this processing block, a computationally simple interpolation algorithm is adopted [4] to provide substantial refinement of the range and azimuth estimations without the need for increasing the range and azimuth FFT sizes.

In the CFAR processing block, we extract k_{peak} in either range or azimuth domain by making it equal to the k index of the largest FFT magnitude sample. Hence, the maximum estimation error in k_{peak} is equal to half the width of the FFT bin. However, if we use the peak sample X_k, and two adjacent samples X_{k-1} and X_{k+1}, the estimate of the peak location can be more accurate if we use simple best- or approximate-fit algorithms.

In the range-azimuth interpolation block, we compute a fractional correction term δ to be added to the integer peak index k to determine a fine estimate of the spectral peak location k_{peak} in the FFT spectrum by $k_{peak} = k + \delta$, as illustrated in Figure 18. Note that δ can be positive or negative, and k_{peak} needs not to be an integer.

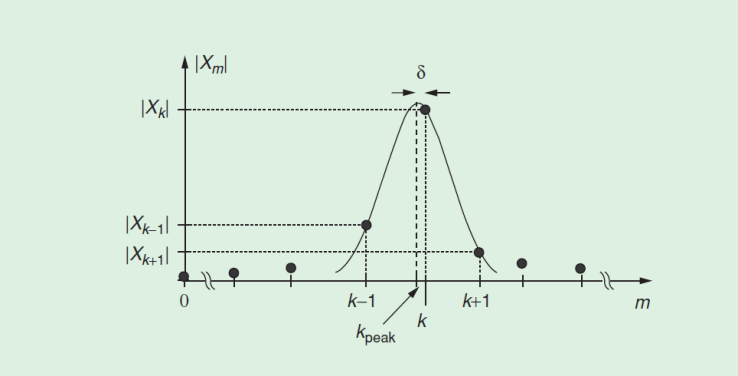


Figure 18: FFT magnitude samples of a spectral tone.

The well-known computationally simple peak location estimation approach fits a parabola through the magnitudes of three interpolation nodes $|X_{k-1}|$, $|X_k|$, and $|X_{k+1}|$ and makes use of a correction term given by

$$\delta = \frac{(|X_{k+1}| - |X_{k-1}|)}{(4|X_k| - 2|X_{k-1}| - 2|X_{k+1}|)} \quad (1)$$

The solution in (1) works well when a rectangular window is applied to the FFT's input samples. As discussed in the angle processing block, no windowing function is applied before the azimuth FFT. Therefore, the interpolation approach in (1) is used in the azimuth domain.

In the range domain, it is often beneficial to use nonrectangular windowing. Therefore, range domain interpolation adopts the computationally simple alternative when data windowing is used

$$\delta = \frac{P(|X_{k+1}| - |X_{k-1}|)}{(|X_k| + |X_{k-1}| + |X_{k+1}|)} \quad (2)$$

where the scaling constant P can be adjusted for different window functions, as listed in Table 1.

Table 1: Correction scaling values for common window types used with (2).

Window	P
Hamming	1.22
Hanning	1.36
Blackman-Harris	1.72

In the current design consideration, an interpolation step is executed on the M4F.

2.6 Motion/Presence Detection

In this processing block, a cuboid-based zone mapping approach is used, and a state machine running parallel for each zone produces a motion/presence decision utilizing the created point cloud set. In the current implementation, the high-level processing blocks are run on the M4F. A portion of these blocks can also be run on the host depending on the use case requirements.

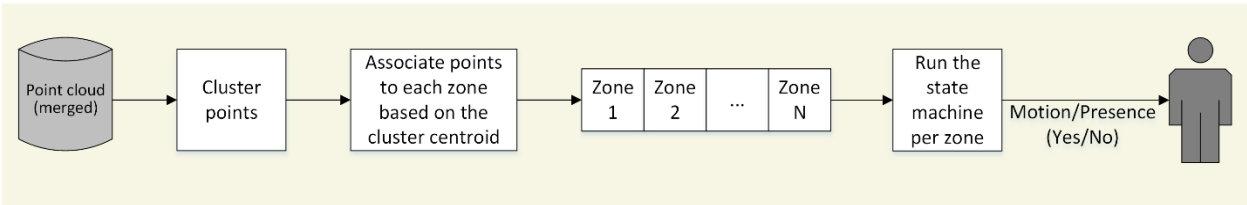


Figure 19: Functional description of high-level processing for motion/presence decision.

In the first step, the detected points are clustered using a DBSCAN algorithm. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points, and D_{ij} be a pairwise Euclidean distance between the item x_i and x_j. The clustering of X is the partitioning of X into K clusters $\{C_1, C_2, \dots, C_K\}$ which satisfy the following conditions:

- Each cluster has at least N points assigned to it, where N can be configured depending on the use case.
- Each point is assigned to one and only one cluster (in the case of hard clustering).

DBSCAN has two important parameters: 'epsilon' and 'minimum points':

- The parameter 'epsilon' defines the radius of the neighborhood around a point in 2D plane.
- The parameter 'minimum points' is the minimum number of neighbor points within the epsilon radius around point x.
- Optionally, total SNR threshold criteria per cluster can also be used in this logic to reduce false alarms caused by multipath, etc.

NOTE: In the current demo implementation, the DBSCAN algorithm only runs on x and y co-ordinates.

After the clustering step, the detected points are mapped into each zone based on the cluster centroids, as illustrated in Figure 20. As depicted, the non-associated points to any cluster are all ignored in the following state machine.

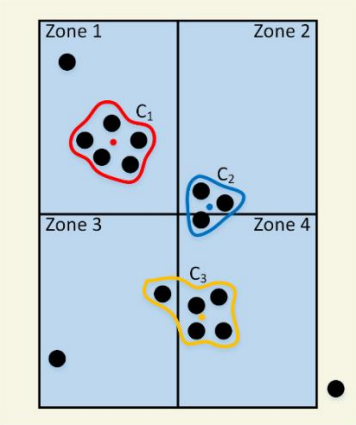


Figure 20: Associating detected points to each zone based on the cluster centroid.

In the last step, a state machine per zone is run to decide the occupancy state of the corresponding zone. To declare a motion/presence in a zone, the following criteria must be satisfied within a point cloud history buffer with a configurable size:

- Points Threshold 1: Minimum number of detected points needed in a zone to enter the motion/presence state.
- Points Threshold 2: Number of detected points needed in a zone to enter the motion/presence state. If the number of points exceeds this threshold, the following average SNR criteria is checked.
- SNR Threshold 2: Total SNR of detected points in a zone to enter the motion/presence state if the 'Points Threshold 2' criteria is also satisfied.

NOTE: In the current implementation of the MPD demo, pointThre1, pointThre2, and snrThre2 are checked when the state goes from empty to minor/major or minor to major. The pointHistThre1, pointHistThre2, and snrHistThre2 are checked when the state goes from major to minor/empty or minor to empty.

Depending on the empirical data analysis for each use case, any of these criteria may be disabled for the sake of simplicity. To enter a motion/presence state from an empty state, the point cloud history buffer size may be set to one (i.e., based on a single frame point cloud set) to reduce the detection latency. On the other, to remain in the motion/presence state, different threshold levels with more buffer sizes can be configured. Similarly, different threshold levels can be used for major and minor point clouds. Finally, to switch from one state to another, some additional temporal filters with configurable time thresholds will improve the robustness. Figure 21 summarizes the state machine running per zone for an example scenario where one of the two zones is occupied.

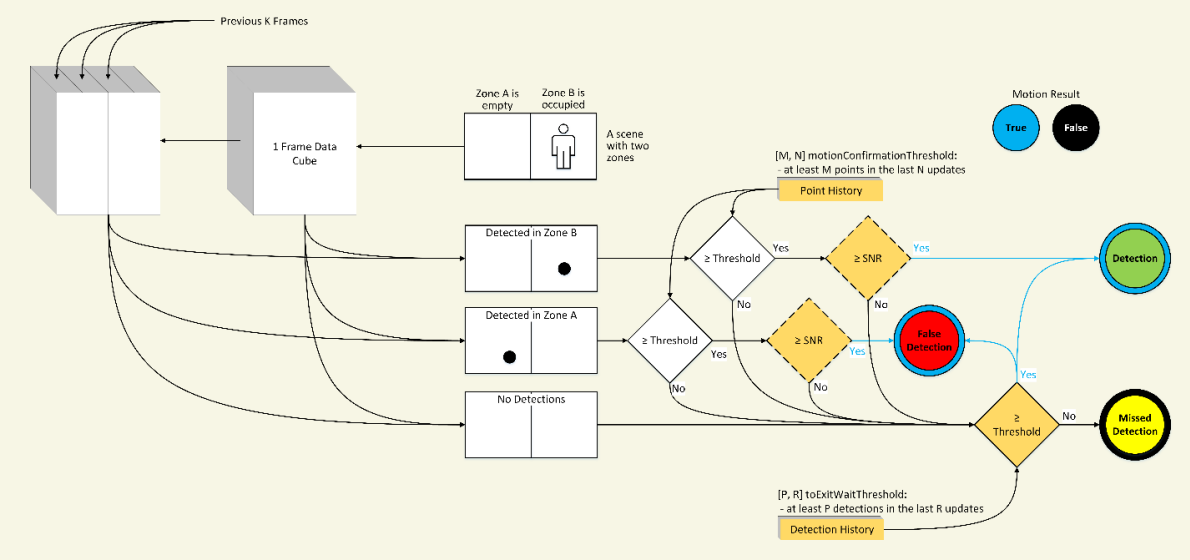


Figure 21: The state machine running per zone based on the point thresholds, SNR thresholds and exit thresholds.

In auto mode (i.e., when both major and minor point clouds are available), the point cloud generated from the current frame is used when searching major motion, and the point cloud generated using multiple frames is used when searching minor motion. In this state machine, the only possible state transition from the empty state is to the major motion state, assuming that the initial motion should be major, as depicted in Figure 22. This approach assumes the chirp blocks within a single frame have enough velocity resolution to detect major motions (walking, etc.) in the scene. When there is a major motion in a target scene, across multiple frames processing may cause more false alarms due to the spreading range and azimuth index over time at low frame rate scenarios. Hence, the state machine in Figure 22 will help to reduce the false alarms.

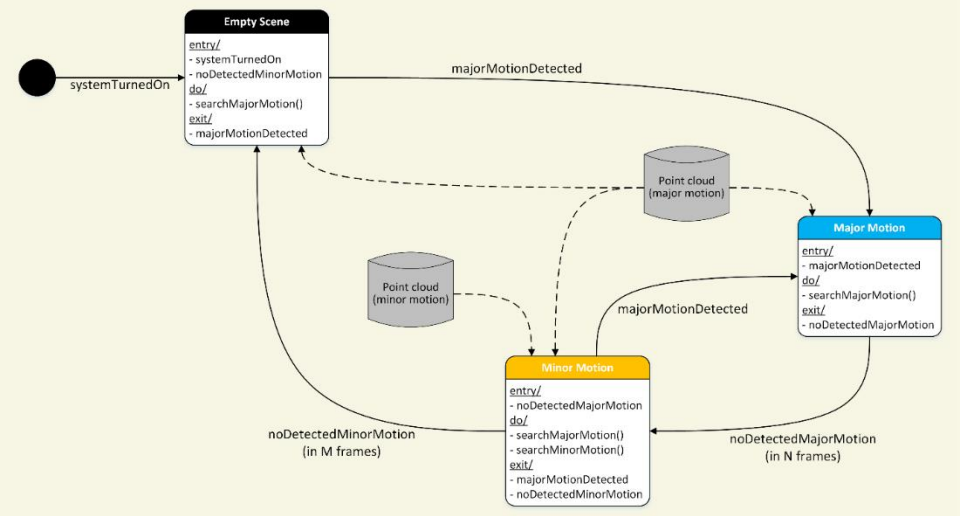


Figure 22: Functional description of the state machine in auto mode.

In some use cases, due to the strict power requirements or limited available memory in larger radar cube sizes, only a single point cloud set (major or minor) can be generated. In this case, the state machine will be simplified, as depicted in Figure 23. Due to the indistinguishable motion type, only two states (empty or motion) will be available and the state transition logic discussed above will be applied on a single point cloud set.

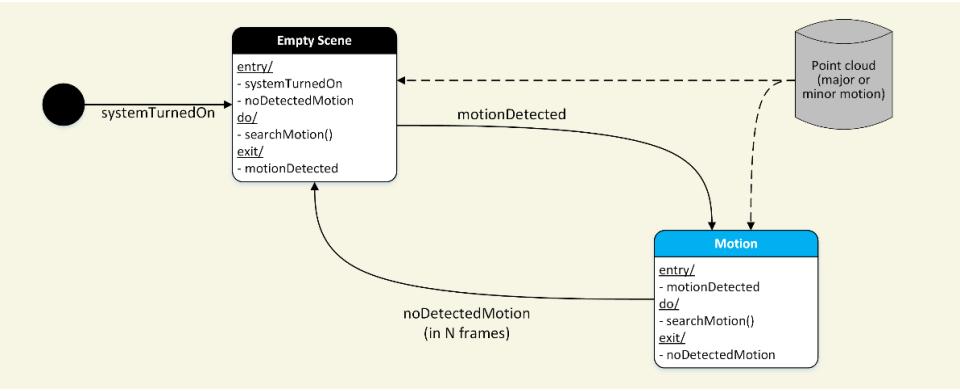


Figure 23: Functional description of the state machine in major or minor only modes.

2.7 Target Tracking

As discussed in Section 1, although the name of the demo is motion/presence detection, the end-to-end processing chain also has the functionality of target tracking and classification. Both high-level processing options are detailed in the following subsections.

As detailed in the previous sections, the detection layer is capable of sensing multiple reflections from real life targets and delivering a rich set of measurement vectors, known as a point cloud. The tracking layer then takes the point cloud data as an input, performs target localization, and reports the results (a target list) for further high-level radar processing (people counting, target classification, etc.). Therefore,

the output of the tracker is a set of trackable objects with certain properties (such as position, velocity, physical dimensions, point density, and other features) that can be used by a higher layer to make further decisions.

In the motion/presence detection demo, the same extended Kalman filter (EKF) based group tracker algorithm in the IWR6843 people counting demo [2] is utilized. The algorithm is integrated to the motion/presence detection demo and tuned to perform successfully with the point cloud generated by the dedicated detection layer discussed in the previous sections. The details of the group tracker algorithm in terms of implementation [2] and tuning [1] are discussed in separate documents. Hence, in this document, we will only summarize the integration-level information to better understand how the same tracker algorithm is being used in the motion/presence detection demo.

As well-known, although it is relatively easy to detect and track people who have enough dynamic motions in the scene, it is challenging to detect and track a static person reliably in a realistic highly cluttered environment. When a person becomes static in the scene, a finer velocity resolution is needed to differentiate between micro-motions on a human body and the fully static clutter. However, due to the limited memory and processing power as well as the required frame rate and power consumption, the velocity resolution, which is a function of the total chirping window, is limited within a single frame. Therefore, as discussed in Section 2.1, to achieve a longer chirping window when detecting minor motions while keeping a robust detection for major motions, it is proposed to create two different chirp blocks in the memory and run the detection layer processing on both chirp blocks in different time slots:

a) The first block uses the available chirps in the current frame with an optimized velocity resolution when detecting the highly dynamic motions,
b) The second block combines the chirp sub-blocks from the current and previous frames to achieve a longer chirping window when detecting a static person in the same scene.

The tracker layer [2] decides the mode (static or dynamic) of a track based on its velocity estimation. It then performs the logic to properly utilize the single or multi frame processing output for dynamic or static tracks. The algorithm flows through the major steps depicted in the block diagram in Figure 24.

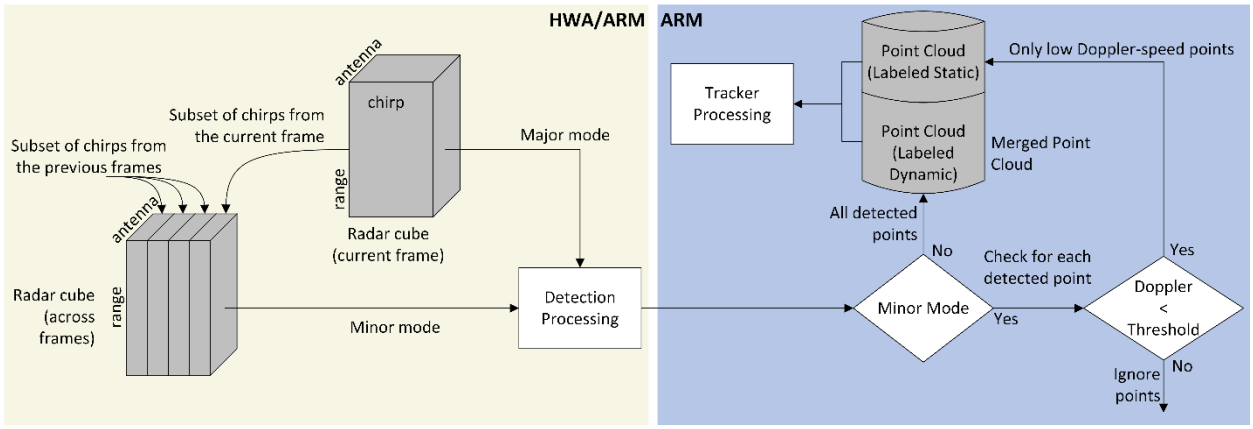


Figure 24: The high-level processing chain to enable tracker in auto (major + minor) mode.

When creating the multi-frame blocks, the number of frames to be combined and the number of chirps selected from each frame should be configured based on a required total observation time window (around 1 second is suggested to observe a typical breathing pattern). Hence, these parameters, which are all software configurable as discussed in Section 3, will depend on the frame rate, the number of chirps available in a single frame, and the velocity resolution required to detect fine motions. Besides, when creating the minor-motion mode radar cube using chirps across multiple frames, the first K chirps per frame is utilized (instead of any subsampling) to keep the maximum unambiguous velocity the same as the single-frame block. It is important to note that the minor-motion mode radar cube should be created with the same size as the single-frame block to fit the rest of the processing chain.

It is obvious that combining multiple frames in minor mode will make the sampling across chirp domain non-uniform. Hence, the conventional FFT processing will prevent accurate Doppler estimations that will eventually confuse the tracker logic. To avoid this, as depicted in Figure 24, the point cloud generated in minor mode is carefully handled with some additional post-processing. When the detection layer runs on the multiple-frames radar cube, the points generated from dynamic tracks are filtered out based on a Doppler threshold (which is software configurable, as discussed in Section 3.2.1). Hence, in the multi-frame processing mode, only the detected points from static targets are desired to be generated to properly run a robust tracker layer logic. Finally, as discussed in [2], to allow the existing group tracker algorithm successfully handling the dynamic and static tracks properly (summarized in Figure 25), the generated point clouds from major and minor mode processing are labeled as dynamic and fully static (zero-Doppler), respectively. In other words, although the points generated in minor-mode (i.e., multi-frame processing mode) has nonzero Doppler (since they are not generated from the pure static clutter), their Doppler will be forced to zero before providing them into the tracker. As discussed in Section 3.2.1, this additional logic is also software configurable through the CLI commands.

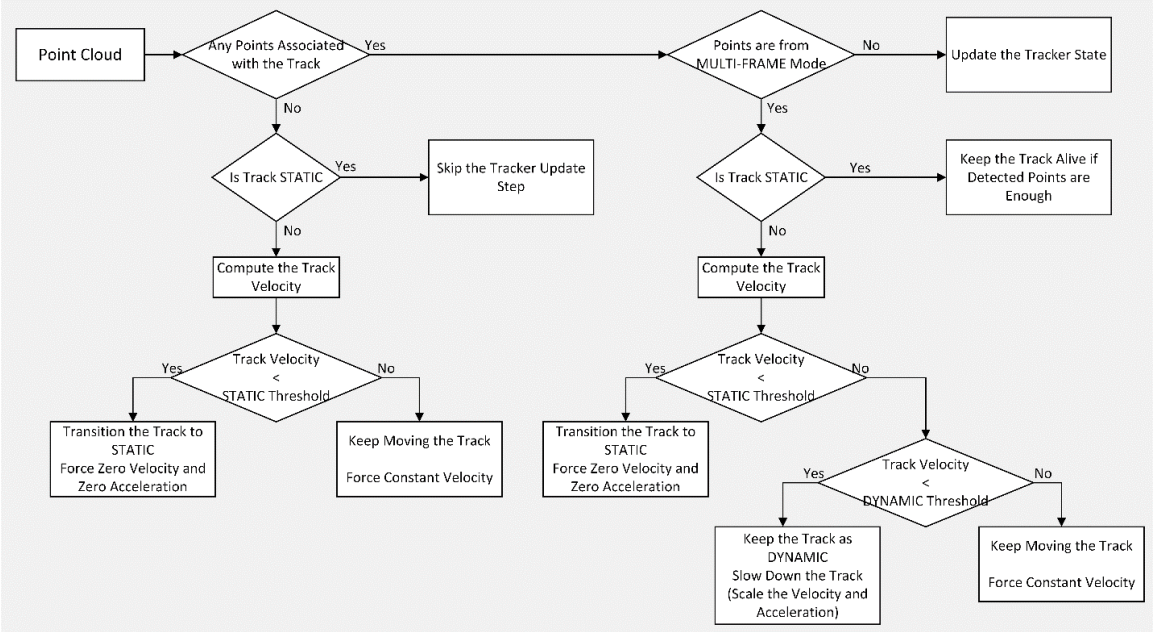


Figure 25: The high-level logic in the tracker layer to handle static and dynamic tracks.

2.8 Target Classification

In this section, we summarize the multiple target classification algorithm [5] we developed in the motion/presence detection demo. In this algorithm, we first create the µ-Doppler spectrogram of each tracked object concurrently utilizing the EKF-based tracking layer integration discussed in the previous section. We then extract 1D time sequence features from the generated µ-Doppler spectrograms and run a pretrained 1D convolutional neural network (CNN) based classifier to address the problem of classifying all the tracked objects as human vs. non-human.

In the high-level processing flow shown in Figure 26, the detection layer (summarized in the previous sections) is first run to generate a point cloud representing the reflections from the scene. On top of the detection layer, the EKF-based group tracker algorithm (see Section 2.7) runs to estimate the centroid of targets and track the motion trajectories of them. While both the point cloud and tracker information are good set of data for sensing, they may not be robust or distinguishable enough in machine learning applications [5]. Hence, the classification algorithm developed in the motion/presence detection demo utilizes the radar cube, which the lowest level of data in our processing chain. As discussed in the following paragraphs, object tracking and spectrogram generation based on referencing the radar cube by the track centroids are central to our approach.

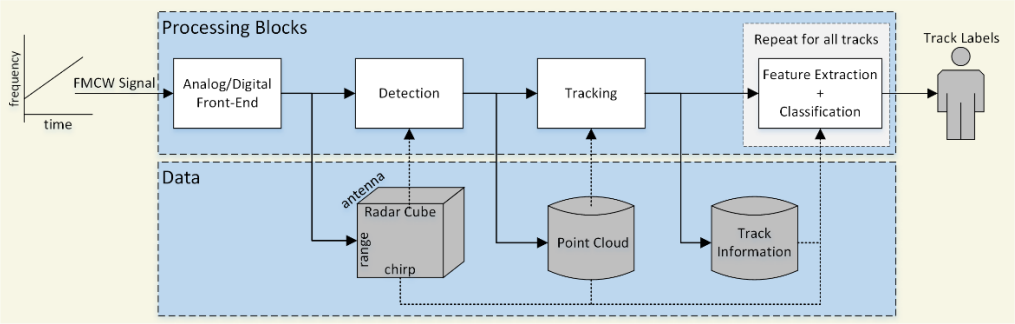


Figure 26: The high-level processing flow that enables the multiple target classification.

In the classification algorithm, a deep-learning based model is built to classify target objects relying on the Doppler domain information (called µ-Doppler spectrograms) extracted directly from the radar cube using the feedback from detection and tracker layers. As summarized in Figure 27, the Doppler spectrum of each pixel around the track is extracted using the target centroid from the tracker. The extracted Doppler spectrum is then projected (by summing across range-angle) into a cumulative µ-Doppler spectrogram of the track. The generated µ-Doppler spectrum per track is then circularly shifted around the corresponding tracker velocity. This processing step is essential to reduce the dependency on numerous possible target scenarios for more robust object classification.

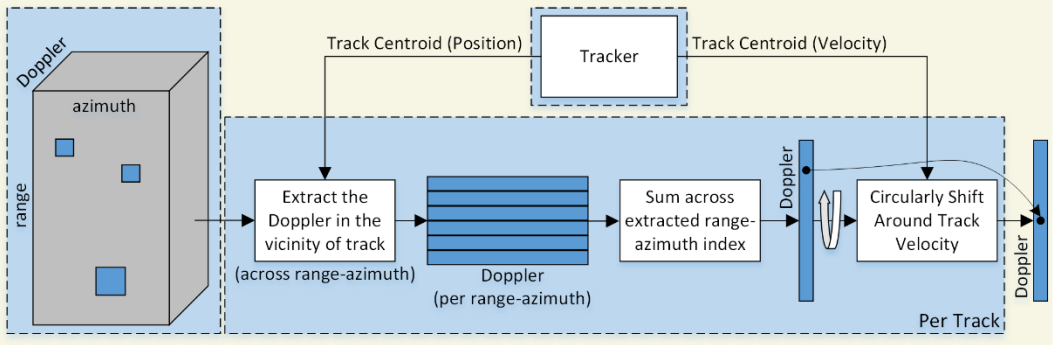


Figure 27: The main processing blocks to generate µ-Doppler spectrograms of tracks.

There are multiple approaches available when creating the µ-Doppler spectrogram per track using the Doppler spectrum around its centroid. The most straightforward method is to extract the Doppler spectrum at the target centroid only and use it at the µ-Doppler spectrogram of that track. In this approach, no accumulation around the track is used, which leads to less SNR and accuracy. In the other approach, the µ-Doppler is extracted by configuring a fixed target size around the centroid and the Doppler spectrums are accumulated within this target boundary. In this method, we fix the range bins (based on the configured track size), and the angle bins are computed adaptively at different ranges (based on the configured track size). In our implementation, any of these methods can be selected through the configuration parameters (will be detailed in Section 3.6.1). For the sake of computational efficiency, the first method (centroid-only) is implemented by Bartlett beamforming (by pointing to a single angle) while the second method adopts the conventional FFT processing in the angle domain (to create multiple angle bins around the track centroid, similar to the detection layer processing in Section 2.4.3).

The generated µ-Doppler spectrograms can be interpreted as 2D images and processed with well-known deep learning techniques (2D-CNN, etc.). In our implementation, to achieve the best performance vs. complexity, we also develop a novel feature extraction method to create 1D time sequence data from the µ-Doppler spectrograms and run a 1D-CNN network, which has much less complexity than 2D-CNN models. In this scope, a total of six similar features from µ-Doppler spectrograms are extracted. To run the 1D-CNN based classification algorithm, the feature extraction is repeated for consecutive frames within a sliding window, whose size is determined based on the tradeoff between the classification accuracy and the latency. In the classifier model built and trained in the demo, a block length of 30 frames (3sec @10Hz) is used. To eliminate the effect of SNR, the µ-Doppler spectrums are normalized between $[0\ 1]$ before the feature extraction step (which can be disabled through the configuration parameters discussed in Section 3.6.1, if different classifier models are desired to be explored).

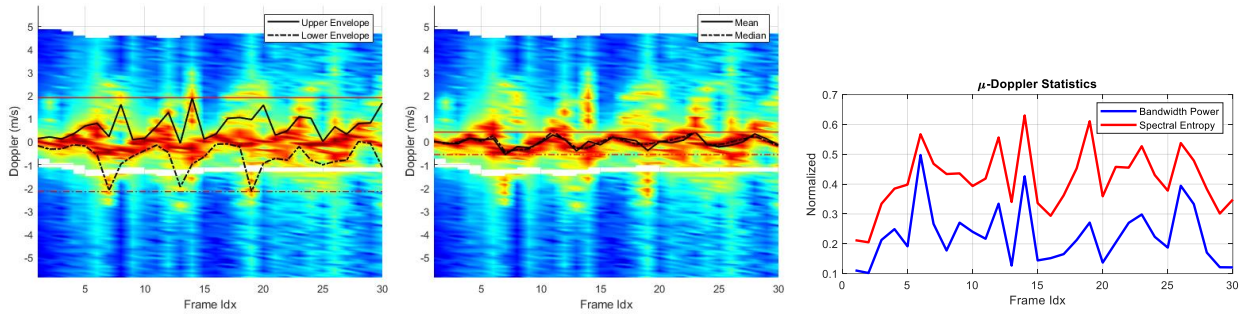
First, the total power of the entire spectrum is computed as $P_T = \sum_n S[n]\Delta_D$, where Δ_D is the Doppler width and $S[n]$ is the spectrum at the nth Doppler bin. The Doppler values that intercept the borders of the spectrum at P_1 and P_2 power ratios (where $P_1 < P_2$) are known as lower (D_{low}) and upper (D_{up}) envelopes and computed by $P_1/100 \times P_T$ and $P_2/100 \times P_T$, respectively. In the computation of upper and lower envelopes, the interception power ratios P_1 and P_2 are heuristically chosen (the suggested

parameters will be discussed in Section 3.6.1). The normalized bandwidth power is extracted as another feature by $P_{BW} = P/100 \times P_T$, where $P = P_2 - P_1$ is the power ratio delta in the configured bandwidth.

The other feature extracted from the µ-Doppler map is called as mean Doppler, which is estimated as $D_{mean} = \left(\sum_n \hat{S}[n]D[n]\right)/P_T$, where $D[n]$ is the Doppler of the nth bin, and $\hat{S}[n] = S[n]\Delta_D$. The median Doppler ($D_{med}$), which is the Doppler value that divides the power equally (i.e., $P_T/2$), is another feature extracted. Finally, the normalized spectral entropy (measure of spectral power distribution) is computed as the last feature by

$$\boldsymbol{H} = -\frac{\sum_{n=1}^{N} P[n]\log_2 P[n]}{\log_2 N}, \qquad \text{where } P[n] = \frac{S[n]}{\sum_i S[i]},$$

where N is the total number of Doppler bins, and $P[n]$ is the probability distribution. In Figure 28, all the time-sequence features generated from an example scenarios are visualized on the µ-Doppler maps.



(a) **(b)** **(c)**

Figure 28: The time-sequence features on µ-Doppler maps: (a) lower and upper envelopes, (b) mean and median, and (c) normalized bandwidth power and spectral entropy.

For robustness in the classification algorithm, some more pre-processing steps are also implemented. If a certain track has no associated points at some frames (it may be obscured by another track), the tracker may still continue to track this object. In such scenarios, we avoid updating the feature set because the extracted information will not be reliable. An example scenario is depicted in Figure 29 to show the corruption in the µ-Doppler map when a person is obscured by trees for a few seconds. When such cases happen, we wait for the next frames to extract new features and delay the classification decision. The main objective behind this approach is; making a wrong decision is worse than making a correct decision with some latency. The minimum number of points required for a track to run the classifier for it is a software configurable parameter discussed in Section 3.6.2.

Note that when working with time-sequence features and networks, the time samples must be in the correct order to allow the classifier better understand the certain signatures in the features. Therefore, we then carefully handle the missing frames (i.e., when a certain track does not have enough associated points) to avoid the sequence being corrupted. To achieve this goal, the index of each frame is kept in the correct order (either filled with fresh data or skipped), and the missing samples are interpolated. It is important to mention that the interpolation has some artifacts we need to avoid to achieve a robust classifier. Hence, in our implementation, the maximum number of consecutive missing frames is limited

to one (i.e., when there is not enough neighbors to enable a simple linear interpolation). As shown in Figure 29, such samples are skipped to refresh the total observation window. Besides, the total number of missing frames in a complete block is also limited (which is configurable as discussed in Section 3.6.2). Finally, extrapolation is avoided at the edges.

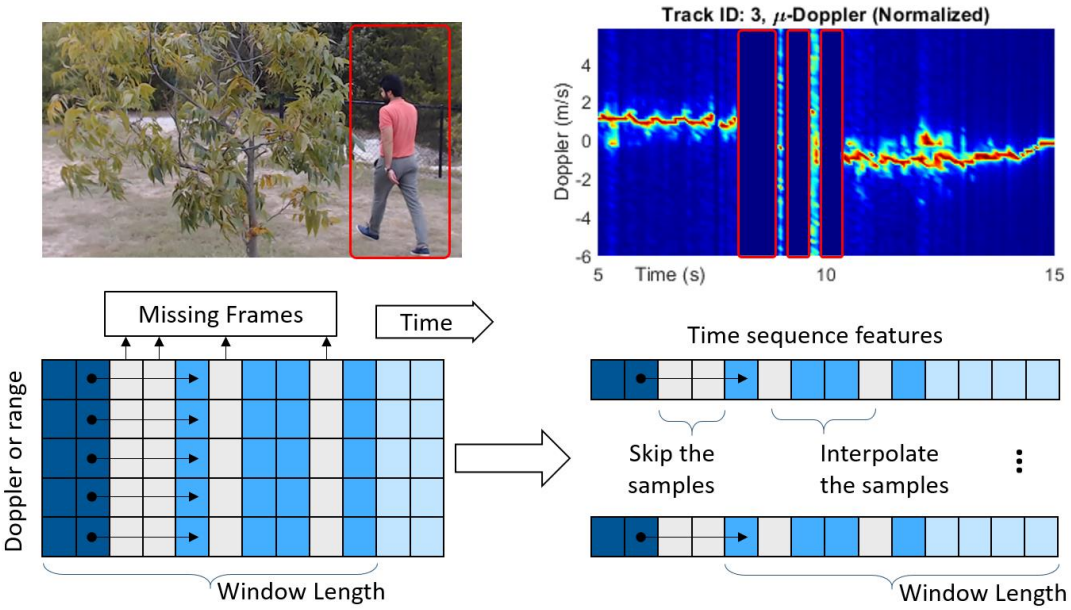


Figure 29: An obscured target scenario to show the corruption in the μ-Doppler map, and the proposed feature extraction approach and interpolation scheme of the missing samples.

As discussed, the μ-Doppler spectrogram is generated around the estimated track centroid. Hence, if the tracker makes an error when estimating that centroid, it will affect the extracted feature set. To improve the robustness, we also adopt the idea of circularly shifting the μ-Doppler spectrum around the mean frequency as depicted in Figure 30. Then the mean (and also median) frequency is no longer used in the feature set of the pretrained classifier. This approach adds slightly more processing in the feature extraction step, but the machine learning model will be simpler and more accurate. But as mentioned before, all the processing options are configurable and more variations can be explored with different classifier and feature generation options.

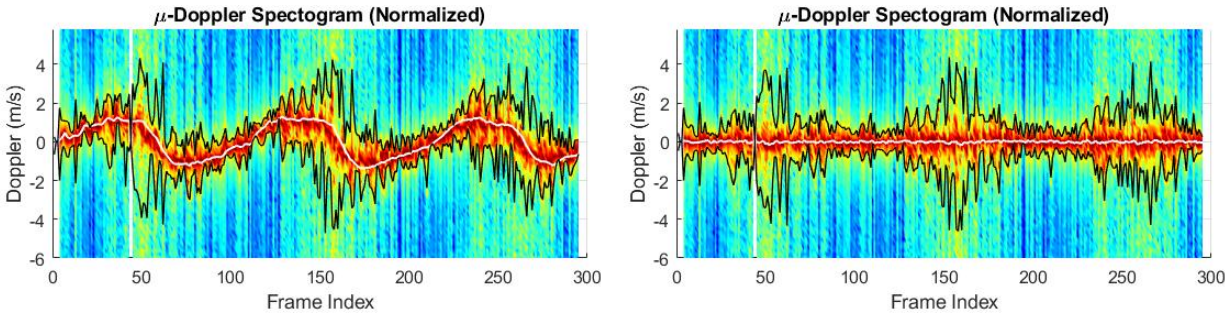


Figure 30: The µ-Doppler spectrum of a human created (a) from the raw measurements and then (b) circularly shifting around the tracker velocity and mean of the spectrum.

As discussed before, we build and train a reference 1D-CNN model to classify the target objects (human or non-human) given the extracted features as time series data. In the proposed 1D-CNN architecture shown in Figure 31, the input size is configured as the total number of features (four by default, excluding the mean and median frequencies as clarified before). Two blocks of 1D convolution, ReLU, and layer normalization layers are specified, where the convolutional layer has a filter size of 3. We create 16 and 32 filters for the first and second convolutional layers, respectively. For both convolutional layers, the inputs are left padded such that the outputs have the same length. To reduce the output of the convolutional layers to a single vector, a 1D global average pooling layer is added. To map the output to a vector of probabilities, a fully connected layer is specified with an output size of two (matching the number of classes), followed by a softmax layer. As illustrated in Figure 31, an optional block of 1D convolution with 64 filters, ReLU, and layer normalization layers (following the first two blocks) is also added to explore a 3 layers CNN model as an option.

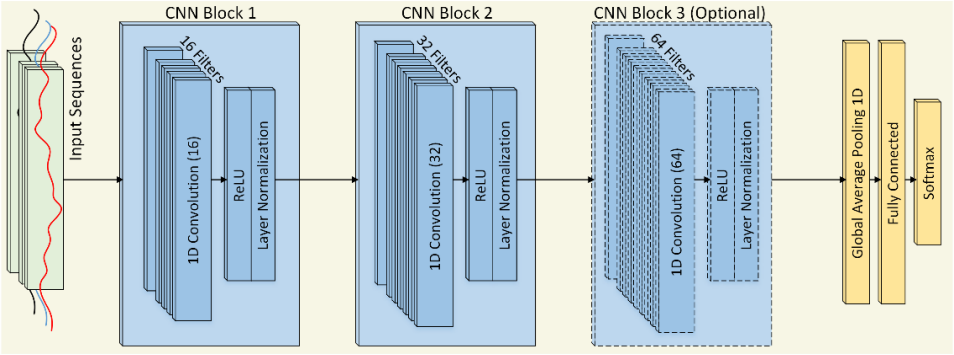


Figure 31: The proposed 2 and 3 layers 1D-CNN architectures.

3 Configuration Parameters

The parameters presented in this section are used to configure the presence/motion detection demo and can be adjusted to match the use cases based on the particular scenery and target characteristics. The user can set the configuration parameters using the configuration (i.e., cfg) files located in the demo application folders. Two example configuration files are given in Table 2 and Table 3 for the minor motion detection use case up to 10m and for the tracking and classification use case in auto mode, respectively. Each line in these files represents a CLI message that configures several parameters. It is important to note that Table 2 or Table 3 may not have the correct sequence of the commands that the demo requires. Besides, there may be some other commands in the cfg files but not listed in these tables. Finally, some of the commands listed in these tables may not be necessary to run the demo for certain use cases. For the complete command set and the correct sequence, please refer to the cfg files located in the demo application folders and the subsequent sections.

Table 2. The commands in an example configuration (cfg) file for the minor motion detection.

No	Parameter group	CLI commands and arguments
1	Sensor front-end parameters	sensorStop 0 baudRate 1250000 channelCfg 7 3 0 chirpComnCfg 20 0 0 64 4 19 0 chirpTimingCfg 6 20 0 32 61 frameCfg 8 0 300 1 250 0 lowPowerCfg 1 factoryCalibCfg 1 0 40 0 0x1ff000 sensorStart 0 0 0 0
2	Detection layer parameters	guiMonitor 2 2 0 0 0 1 1 0 0 0 0 sigProcChainCfg 64 2 2 0 4 4 0 20 cfarCfg 2 4 3 2 0 12.0 0 0.5 0 1 1 1 aoaFovCfg -70 70 -40 40 rangeSelCfg 0.1 10.0 clutterRemoval 1 antGeometryCfg 0 0 1 1 0 2 0 1 1 2 0 3 2.418 2.418 compRangeBiasAndRxChanPhase 0.0 1 0 -1 0 1 0 -1 0 1 0 -1 0 measureRangeBiasAndRxChanPhase 0 2 0.2
3	Debug related parameters	adcDataSource 0 adc_data_name.bin adcLogging 0
4	Motion/presence detection layer parameters	mpdBoundaryBox 1 -5 5 0 10 0 3 sensorPosition 0 0 2 0 0 minorStateCfg 4 3 10 8 6 20 8 20 clusterCfg 1 0.5 2

Table 3. The commands in an example configuration (cfg) file for tracking and classification.

No	Parameter group	CLI commands and arguments
1	Sensor front-end parameters	sensorStop 0 baudRate 1250000 channelCfg 7 3 0 chirpComnCfg 16 0 0 128 4 28 0 chirpTimingCfg 6 32 0 40 60.5 frameCfg 2 0 200 64 100 0 lowPowerCfg 0 factoryCalibCfg 1 0 40 0 0x1ff000 sensorStart 0 0 0 0
2	Detection layer parameters	guiMonitor 2 3 0 0 0 1 0 0 1 1 1 sigProcChainCfg 32 2 3 2 8 8 1 0.3 cfarCfg 2 8 4 3 0 10.0 0 0.5 0 1 1 1 aoaFovCfg -70 70 -40 40 rangeSelCfg 0.1 10.0 clutterRemoval 1 antGeometryCfg 0 0 1 1 0 2 0 1 1 2 0 3 2.418 2.418 compRangeBiasAndRxChanPhase 0.0 1 0 -1 0 1 0 -1 0 1 0 -1 0 measureRangeBiasAndRxChanPhase 0 2 0.2
3	Debug related parameters	adcDataSource 0 adc_data_name.bin adcLogging 0
4	Tracking layer parameters	boundaryBox -3.5 3.5 0 9 -0.5 3 sensorPosition 0 0 1.9 0 15 staticBoundaryBox -3 3 0.5 7.5 0 3 gatingParam 3 2 2 2 4 stateParam 3 3 12 50 5 200 allocationParam 6 10 0.1 4 0.5 20 maxAcceleration 0.4 0.4 0.1 trackingCfg 1 2 100 3 61.4 191.8 100 presenceBoundaryBox -3 3 0.5 7.5 0 3
5	Classification layer parameters	microDopplerCfg 1 0 0.5 0 1 1 12.5 87.5 1 classifierCfg 1 3 4

If the trackingCfg parameter is present and enabled(along with other relevant parameters) in the user-provided configuration, then the demo runs on the tracker chain. However, if the minorStateCfg /majorStateCfg parameter is present, then the demo runs on the MPD chain. It is important to note that only one set of configs/processing (MPD chain/trac

ker chain) is supported at a time by the demo, and both cannot be run together for a given frame.

This document elaborates specifically on the supported configuration parameters in the motion/presence detection demo. The remaining parameters (and the configuration options they provide) not detailed in this document (at the sensor front-end level) are described in the ICD [6]. In this document, we also briefly review these parameters and discuss their effect and limitations in the overall motion/presence detection

demo. It is important to emphasize that we recommend using the provided values of the configuration parameters as the best empirical values we obtained from testing in different environments. In the following sections, we provide detailed information on each parameter and discuss their effect on the detection performance for different scenarios.

The first table below summarizes the default commands, which do not need a specific tuning effort, to configure the RF module parameters of the radar sensor, such as enabling the transmit and receive antennas, start and stop the sensor, etc.

Commands	Parameters	Usage in demo	Notes
sensorStop	frameStopMode	Set to 0	Indicates the frame stop mode. This command stops the sensor from transmitting further frames and is not supported when lowPowerCfg is 1 (see the following sections). The recommended alternative is to power cycle for a reconfiguration.
baudRate	baudRate	Set to 1250000 for the full UART speed.	The sensor starts with 115200 baud rate by default. When this command is sent to the device, the baud rate for the UART communication is updated according to the given value.
channelCfg	rxChCtrlBitMask	Set to 7 to enable all available RX antennas.	To enable (as an example): - RX antennas 1 and 2, mask = 0x011b = 3 - RX antennas 1 and 3, mask = 0x101b = 5 - RX antennas 2 and 3, mask = 0x110b = 6 - RX antennas 1, 2 & 3, mask = 0x111b = 7
	txChCtrlBitMask	Set to 3 to enable all available TX antennas.	To enable: - TX antennas 1 and 2, mask = 0x011b = 3 - TX antenna 1, mask = 0x001b = 1 - TX antenna 2, mask = 0x010b = 2
	miscCtrl	Set to 0	Not supported on the current version of the SDK.
sensorStart	frameTrigMode	Set to 0	Always set to 0 in the current version of the SDK to enable frame SW immediate trigger mode [6].
	chirpStartSigLbEn	Set to 0	Always set to 0 in the current version of the SDK.
	frameLivMonEn	Set to 0	When logging ADC data from DCA, set it to 2 if side band ADC data is needed.
	frameTrigTimerVal	Set to 0	Always set to 0 in the current version of the SDK.

3.1 Sensor Front-End Parameters

The sensor front-end parameters, which configure the motion/presence detection demo at the frequency modulated continuous wave (FMCW) signal level, are given in this section. The following subsections briefly summarize these parameters at the demo level to provide a complete guide for the user. To get a general understanding of the sensor front-end parameters, please refer to the SDK [1] and ICD [6].

3.1.1 Example Chirp Configurations

In Table 4, some example chirp configurations are provided for motion/presence demo to support multiple range requirements and different use cases. The descriptions of the parameters in Table 4 and the relations between these parameters and the commands in the configuration files will be discussed in the following subsections.

According to the existing FCC regulations (subject to change), if only the ISM band is utilized (<500MHz), full available TX power can be used. Otherwise, the TX power should be limited. Hence, due to the long-range requirements given for the motion detection use cases, low-bandwidth chirp configurations can be used to utilize the full available TX power at longer ranges. Note that the occupied spectrum in the low-bandwidth motion detection configurations is configured for demo purposes. The start frequency in Table 4 can be adjusted according to the available bands in real-world applications. In these configurations, the chirping parameters, such as slope, idle time, ADC start time, etc., can also be tuned better to achieve better performance (power consumption, bandwidth, etc.).

Table 4: Example chirp configurations for the motion/presence demo.

		High Bandwidth				Low Bandwidth	
Parameters	**Units**	**2m**	**3.7m**	**6m**	**10m**	**10m**	**15m**
Starting frequency	GHz	59.7	59.75	60	60	61	61
Ramp slope	MHz/µs	120	90	74	75	32	19
# of ADC samples per chirp	-	64	128	128	256	64	128
# of chirp loops (per TX antenna)	-	2	4	4	64	4	32
ADC sampling rate decimator	-	25	20	15	8	20	20
ADC sampling rate	Msps	4	5	6.67	12.5	5	5
Idle time for TX1	µs	6	6	6	138	6	230
Idle time for TX2	µs	6	6	6	6	6	6
Chirp repetition period (T_r)	µs	51.0	69.6	55.8	189.6	44.0	293.6
Filter type (long or short)	-	Long	Long	Long	Long	Long	Long
# of ADC skip samples	-	22	24	26	37	24	24
Filter group delay (long filter: 12 samples, short filter: 8 samples)	us	3	2.4	1.8	0.96	2.4	2.4
Equivalent ADC start time	µs	2.5	2.4	2.1	2	2.4	2.4
Ramp end time	µs	19.5	28.8	21.9	22.8	16	28.8
Frame period	ms	500	250	250	100	250	250
Chirp time	µs	16.00	25.60	19.20	20.48	12.80	25.60
Valid bandwidth	MHz	1920	2304	1421	1536	410	486
Total bandwidth	MHz	2340	2592	1620.6	1710	512	547.2
Carrier frequency	GHz	60.96	61.12	60.87	60.92	61.28	61.29
Maximum unambiguous range	m	2.25	3.75	6.08	11.25	10.55	17.76
Range resolution	cm	7.81	6.51	10.56	9.77	36.59	30.86
Maximum unambiguous velocity	m/s	24.12	17.63	22.08	6.49	27.81	4.17

Velocity resolution	m/s	24.12	8.82	11.04	0.20	13.91	0.26

In Table 4, some example chirp configurations are also provided with high-bandwidth options. When high-bandwidth configurations are needed to achieve better range resolutions, these chirp configurations can be considered. To meet the available FCC regulations, TX power can be limited using the TX backoff feature of the xWRLx432 (refer to Section 3.1.5). When the FCC regulations are waived in any use case, these chirping schemes can be considered with full available TX power.

It is also important to note that the maximum unambiguous velocities and velocity resolutions in some of these configurations are not optimal for estimating the correct Doppler velocity of a typical human motion or resolving a typical human motion from the static clutter. In the motion/presence detection use case, due to the strict power consumption requirements, the target velocity is not used in the default motion/presence decision logic. On the other hand, if the target velocity is needed by sacrificing the average power consumption (e.g. for target tracking and/or classification), a greater number of chirps can be considered. Besides, when the minor mode is enabled, the chirps are closely grouped in a single frame (which results in lower velocity resolution). As discussed in Section 2.1, to handle the motion/presence detection with the given chirp configurations, multiple frames will be combined to increase the effective chirping window (i.e., minor mode).

To achieve enough velocity resolution to detect major motions within a single frame (i.e., major mode), a long chirp repetition period should be configured. In this scheme, two chirps per MIMO scheme (either BPM or TDM) should be close to each other in the time domain (i.e., a short idle time 2 in Figure 8) to reduce the angle estimation errors caused by the phase difference across transmit antennas. On the other hand, a longer idle time between two consecutive chirp pairs can be used (i.e., idle time 1 in Figure 8) to improve the velocity resolution.

3.1.2 Chirp Profile Configuration

We develop the motion/presence detection signal processing chain to run on the TI's mmWave sensors that use the FMCW (also known as chirp) signal waveform. The FMCW chirp configuration is an important performance aspect that needs to be considered in the motion/presence detection chain. The basic principles of FMCW radars are well-reported in the literature [7][8]. The xWRLx432 mmWave sensors provide flexibility in configuring the chirp waveforms through the following chirpComnCfg and chirpTimingCfg commands. This section summarizes the parameters in these commands that govern the chirp configurations optimized for the motion/presence detection demo. The parameters given in these commands are common for all chirps in a frame.

Note that the tables given in the rest of the document present the data type and description of each parameter along with the suggested values in an example scenario (the minor motion detection use case up to 10m with a low-bandwidth option). The data types are only classified as ‘int’ for integer numbers and ‘float’ for floating-point numbers. This document does not cover the actual byte size of the parameters (uint8, int32, etc.). Instead, the supported values or valid ranges in each parameter are given

in the corresponding section. For more details about the actual data size, please refer to the SDK [1] and ICD [6].

chirpComnCfg 20 0 0 64 4 19 0			
Parameter	**Type**	**Value**	**Description**
digOutputSampRate	int	20	Sampling rate decimator. The digital sampling rate is given by 100MHz/digOutputSampRate. Valid Range: 8 to 100. Supported sampling rates: 1.0, 1.25, 1.667, 2.0, 2.5, 4.0, 5.0, 6.667, 7.692, 10.0, 12.5Msps. Refer to [8] for the details about the sampling rate.
digOutputBitsSel	int	0	Digital output sample bits select. Refer to the ICD [6] and the note below[1] for the details.
dfeFirSel	int	0	The final stage FIR filter's characteristics. Refer to the ICD [6] and the note below[2] for the details.
numOfAdcSamples	int	64	Number of ADC samples collected during the ADC sampling time. Current SDK works for values with powers of 2. Valid Range: 2 to 2048. Refer to [8] for the details.
chirpTxMimoPatSel	int	4	Configure BPM/TDM mode. If 1 TX is enabled: Set to 0 to disable the multiplexing mode. If 2 TX is enabled: Set to 1 to enable TDM mode Set to 4 to enable BPM mode. Suggested to set 4. Refer to the note below[3] for the details.
chirpRampEndTime	float	19	Ramp end time in µs. Refer to [8] and the description below[4] for the details.
chirpRxHpfSel	int	0	Chirp profile HPF corner frequency. Refer to the ICD [6] and the note below[5] for the details.

[1]**digOutputBitsSel:** This field governs which bits of the FECSS DFE's internal 16-bit signed data path are sent as output. The digital filter chain output is always sent as a 16-bit bus per RX. In the cases where only 12 bits are sent, they are sent on the LSBs, and the remaining 4 MSBs in the interface are sign extended. The output bit selection options are as below:

- 0 - Digital sample output is 12 MSB bits of DFE after rounding 4 LSBs.
- 1 - Digital sample output is 12 bits after rounding 3 LSBs & clipping 1 MSB.
- 2 - Digital sample output is 12 bits after rounding 2 LSBs & clipping 2 MSB.
- 3 - Digital sample output is 12 bits after rounding 1 LSBs & clipping 3 MSB.
- 4 - Digital sample output is 12 LSB bits after clipping 4 MSB.
- 5 - Digital sample output is 16 bits.

[2]**dfeFirSel:** The final stage FIR filter's characteristics can be selected as below:

- 0 - Long Filter (90% visibility): This provides visibility to a larger range of IF frequencies: 0 to 0.45 x Sampling Rate. Beyond that, the filter response starts dropping and enters the filter transition band.
- 1 - Short Filter (80% visibility): This provides faster settled outputs, but the IF frequency range visible is 0 to 0.40 x Sampling Rate. Beyond that, the filter response starts dropping and enters filter transition band.

[3]**chirpTxMimoPatSel:** To enable the MIMO mode on the mmWave sensors [7], the receiving elements must be able to separate the signals corresponding to different transmitter antennas. In the motion/presence detection demo, the orthogonality between the transmit antennas is achieved by employing either TDM or BPM techniques, as discussed in Section 2.3.1. This parameter configures the TX antenna multiplexing scheme (BPM or TDM) to enable the MIMO mode on the sensors. In BPM-MIMO mode, with the simultaneous transmission on both TX antennas, the total transmitted power per chirp interval is increased, which translates to an SNR improvement of 3dB. Hence, in the motion/presence detection demo, it is suggested to use BPM-MIMO mode to improve the detection performance. On the other hand, the average power consumption also increases with the BPM-MIMO mode compared to the TDM-MIMO scheme.

[4]**chirpRampEndTime:** This parameter is the sum of the ADC start time (refer to chirpAdcSkipSamples parameter below), ADC sampling time, and some ramp excess time. The user should program Ramp End Time higher than the time necessary for RX ADC sampling and filtering. The time necessary is a function of the sampling rate, Fs, in Hz, determined by digOutputSampRate, as well as the number of skip samples (chirpAdcSkipSamples) and collect samples (numOfAdcSamples). Specifically,

RX sampling rate (Fs)	dfeFirSel	Ramp End Time *(in seconds)*
< 1.5 MHz	0 (long filter)	Ramp End Time ≥ (Skip Samples + Collect Samples – 4) /Fs *seconds*
	1 (short filter)	
≥ 1.5 MHz	0 (long filter)	Ramp End Time ≥ (Skip Samples + Collect Samples – 8) /Fs *seconds*
	1 (short filter)	Ramp End Time ≥ (Skip Samples + Collect Samples – 5) /Fs *seconds*

Note: There is a group delay incurred in the RX baseband filtering (for skipping initial samples and collecting the required number of samples). The filter group delay is just a "pipeline latency" and doesn't require expanded ramp end time. The above table uses this fact to optimize the minimum ramp end time constraint.

In setting the chirp timing parameters and sample counts, the user should ensure that the programmed chirp periodicity, i.e., Ramp End Time + Idle Time (set by fields: chirpRampEndTime, chirpIdleTime) always exceeds the time necessary for RX ADC sampling. Specifically, ensure that

(Ramp End Time + Idle Time) ≥ (Skip Samples + Collect Samples + 4) / Fs

[5]**chirpRxHpfSel:** Chirp profile HPF corner frequency. The supported options are given below:

- 0 - 175kHz HPF corner frequency

- 1 - 350kHz HPF corner frequency
- 2 - 700kHz HPF corner frequency
- 3 - 1400kHz HPF corner frequency

chirpTimingCfg 6 20 0 32 61			
Parameter	**Type**	**Value**	**Description**
chirpIdleTime	float	6	Idle time in µs. Refer to [8] and the note below[1] for the details.
chirpAdcSkipSamples	int	20	Number of samples skipped during ADC sampling. The ADC start time in µs is computed based on the sampling rate. Valid range: 0 to 63. Refer to [8] and the note below[2] for the details.
chirpTxStartTime	float	0	TX start time in µs. Recommended Range: +/- 5µs. Refer to [8] and the note below[1] for the details.
chirpRfFreqSlope	float	32	Frequency slope in MHz/µs. Valid range: - 399MHz/µs to +399MHz/µs. Refer to [8] for the details.
chirpRfFreqStart	float	61	Chirp profile RF start frequency in GHz. Valid range: 58GHz to 62.5GHz for IWRL6432 devices. Refer to [8] and the note below[3] for the details.

[1]**chirpIdleTime and chirpTxStartTime:** The parameter chirpIdleTime indicates the idle time in the chirp cycle before starting of the frequency ramp. Along with the ramp end time, the number of TX antennas, and the number of chirp loops in a frame (as discussed in Section 3.1.3), this parameter configures the motion sensitivity (i.e., velocity resolution) of the detection layer (please refer to Section 4 for details). The user can adjust this parameter according to desired motion sensitivity in the major motion processing mode (as discussed in Section 2.1). It is important to note that the user must ensure that the total chirping time and the required processing time can fit into a single frame period, which is discussed in Section 3.1.3. The parameter chirpTxStartTime indicates the TX start time in the chirp cycle with respect to the knee of the ramp.

- If chirpIdleTime is >= maximum of {6µs - chirpTxStartTime, 3.1µs, (1+ RF_Bandwidth/1400MHz) in µs} then the SDK code enables inter-chirp power save for TX and RX chains.
- Else-if chirpIdleTime is >= the maximum of {4µs - chirpTxStartTime, 3.1µs, (1+ RF_Bandwidth/1400MHz) in µs} then the inter-chirp power save mode is enabled for TX chains but disabled for RX chains.
- Else, it will throw an error. Therefore, the chirpIdleTime shouldn't be configured lower than maximum of {4µs - chirpTxStartTime, 3.1µs, (1+ RF_Bandwidth/1400MHz) in µs}.

- In above formulae, the RF_Bandwidth is calculated as the product of abs(chirpRfFreqSlope) and chirpRampEndTime.

[2]**chirpAdcSkipSamples:** The parameter configures the number of ADC samples to be skipped at the knee of the ramp, which controls the ADC sampling start time. Note that the term "skip samples" refers to the number of ADC samples at the beginning of each chirp which are skipped (i.e. not collected for radar processing). In each chirp, the initial samples exhibit settling behavior, which arises in the synthesizer PLL and RX baseband high-pass and low-pass filters (analog and digital). The skip samples parameter needs to be high enough to wait out these (potentially abrupt) initial transients. Specifically, the recommended parameters are given as:

Chirp start freq. (60GHz devices)	Chirp start freq. (77GHz devices)	c_DfeFirSel	Skip Samples Recommended
[57.5, 64] GHz	[76.5, 81] GHz	0 (long filter)	Num Skip Samples ≥ 12 + max [(chirpTxStartTime + 1.5us)*Fs + 4, (2us*Fs)] *samples*
		1 (short filter)	Num Skip Samples ≥ 08 + max [(chirpTxStartTime + 1.5us)*Fs + 3, (2us*Fs)] *samples*
Otherwise	Otherwise	0 (long filter)	Num Skip Samples ≥ 12 + max [(chirpTxStartTime + 1.5us)*Fs + 4, (3us*Fs)] *samples*
		1 (short filter)	Num Skip Samples ≥ 08 + max [(chirpTxStartTime + 1.5us)*Fs + 3, (3us*Fs)] *samples*

Note: the 1.5µs in the above table corresponds to the recommended HPF Fast Init Duration (refer to the ICD [6]). The other parameters correspond to the group delay and settling delay of LPFs (which depend on the user's filter type selection) and synthesizer PLL settling delay (which depends on RF start frequency (see chirpRfFreqStart)).

[3]**chirpRfFreqStart:** This parameter is the start frequency of the total occupied bandwidth. Along with the frequency slope and the ADC start/sampling time, this parameter configures the valid bandwidth occupied in the raw FMCW beat signal to be utilized in the detection layer. The given values for the presence/motion detection demo are configured empirically for the best performance according to the radiation patterns of the xWRLx432 EVM antennas. The user might need to adjust these values for their customized antenna designs.

3.1.3 Frame Configuration

Finally, the following command configures the FMCW radar frame timing parameters like num of chirps, burst and frame counts, burst & frame periodicity, etc. For details about a typical frame structure (chirps, bursts, and frames), please refer to the ICD [6] and SDK [1].

frameCfg 8 0 300 1 250 0			
Parameter	**Type**	**Value**	**Description**
numOfChirpsInBurst	int	8	Number of chirps in a burst. Valid Range: 1 to 65535 chirps. Refer to the note below[1] for the details.
numOfChirpsAccum	int	0	Number of accumulations per chirp. 0: Chirp accumulation feature is disabled. 2 – 64: Chirp accumulation feature is enabled.

			Refer to the note below[2] for the details.
burstPeriodicity	float	300	Burst periodicity in µs. Refer to the note below[1] for the details.
numOfBurstsInFrame	int	1	Number of bursts per frame. Valid range: 1 to 4096. Refer to the note below[1] for the details.
framePeriodicity	float	250	Frame periodicity (i.e., the time gap between successive frame starts) in ms. Refer to the note below[3] for the details.
numOfFrames	int	0	Number of total frames. 0: Infinite frames, value > 0: Finite frames. Valid Range: 0 to 65535. Always set to 0 to run the demo in continuous mode (i.e., to generate endless frames).

[1]**numOfChirpsInBurst, burstPeriodicity, and numOfBurstsInFrame:** These parameters configure the total number of chirps transmitted within a single frame. Along with the chirp timing parameters in Section 3.1.2, these parameters configure the motion sensitivity (i.e., velocity resolution) of the detection layer and the maximum detection range based on SNR (please refer to Section 4 for details). These parameters affect the radar cube size, required processing time, and average power consumption, which should be carefully optimized for the desired use case.

As discussed in the SDK [1], although the digital front-end has the flexibility of configuring any combination of chirps/bursts, the motion/presence detection demo only supports two possible options:

- Normal Mode: The numOfBurstsInFrame is configured to 1, and the number of chirps desired in a frame is configured by numOfChirpsInBurst. It is important to note that the numOfChirpsInBurst is the total number of chirps available (from both TXs). For example, if 32 chirps per TX is desired and if both TXs are enabled, then the numOfChirpsInBurst should be configured to 64 in this mode.
- Burst Mode: The numOfBurstsInFrame can be configured to >1, but the numOfChirpsInBurst must be configured to 1 (when only 1 TX is enabled) or 2 (when both TXs are enabled). Then the numOfBurstsInFrame will correspond the total number of chirps in a frame per TX antenna.

Below snippets from ICD [6] cover the burst and frame time constraints.

Variable	Description	xWRL6432 xWRL1432
Burst Start Time $BurstStartTime$	Time required for RFS firmware to process the Burst Start event	65 us
Burst End Time $BurstEndTime$	Time required for RFS firmware to process the Burst End event	50 us
Minimum Burst Idle Time $MinBurstIdleTime$	$BurstStartTime + BurstEndTime$	115us

Variable	Description	xWRL6432 xWRL1432
Frame Start Time $FrameStartTime$	Time required for RFS firmware to process the Frame Start event	25 us
Frame End Time $FrameEndTime$	Time required for RFS firmware to process the Frame End event	15 us
Minimum Frame Idle Time $MinFrameIdleTime$	$FrameStartTime + FrameEndTime$	40 us

[2]**numOfChirpsAccum:** This field indicates the number of chirps to be accumulated before sending the ADC data out in DFE. This feature can be used to increase the SNR without increasing the number of chirps to process in the DSP/HW accelerator. When this mode is enabled, the front-end device will send more chirps than in the processing layer (since they are accumulated before filling the ADC buffer), which will increase power consumption. Hence, this parameter should be carefully optimized for the desired use case and power consumption.

[3]**framePeriodicity:** This parameter configures the frame rate of the processing chain. The user must always ensure that the total chirping time and the required processing time can fit into a single frame period. This parameter also affects the average power consumption. The framePeriodicity should be configured to allow a minimum 135 µs inter-frame idle time.

3.1.4 Low-Power Mode Configuration

The following command enables/disables the low-power mode. This command is used to enable/disable various sensor inter-chirp and inter-burst dynamic power save options.

lowPowerCfg 1			
Parameter	**Type**	**Value**	**Description**
enable	int	1	Low power framework enable/disable flag: 0: Disabled. 1: Enabled. Refer to the ICD [6] for the details.

3.1.5 Factory Calibration Configuration

Factory calibration save/restore command. Provides the user to either save the factory RF calibration performed by the FECSS onto the FLASH or to restore the previously saved RF calibration data from the FLASH and instruct FECSS not to re-perform the factory calibration. Users can either save or restore or perform neither operation. The user is not allowed to save and restore in a given boot sequence simultaneously. NOTE: Factory Calibration CLI command should be issued within 25°C +/- 15°C.

factoryCalibCfg 1 0 40 0 0x1ff000

Parameter	Type	Value	Description
saveEnable	int	1	Save calibration enable/disable flag: 0: Save disabled. 1: Save enabled. Refer to the note below[1] for the details.
restoreEnable	int	0	Restore calibration enable/disable flag: 0: Restore disabled. 1: Restore enabled. Refer to the note below[2] for the details.
rxGain	int	40	RX gain in dB scale. Recommended Range: Even values from 30dB to 40dB.
txBackoff	int	0	TX channel power calibration. Valid Range: 0dB to 26dB for xWRL6432, 0dB to 20dB for xWRL1432. Refer to the note below[3] for the details.
flashOffset	int	0x1ff000	Always set to 0x1ff000. Refer to the note below[4] for the details.

[1]**saveEnable:** When this option is enabled application will boot-up normally and configure the FECSS to perform all applicable factory calibrations during FECSS initialization. Once the calibrations are performed, the application will retrieve the calibration data from FECSS and save it to FLASH. Users need to specify a valid flashOffset value. Besides, restoreEnable option should be set to 0.

[2]**restoreEnable:** When the restore enabled option is set, the application will check the FLASH for a valid calibration data section. If present, it will restore the data from FLASH and provide it to FECSS while configuring it to skip any real-time factory calibrations and use provided calibration data. Users need to specify the valid flashOffset value that was used during the saving of calibration data. Besides, saveEnable option should be set to 0. The rxGain and txBackoff arguments will be ignored when the restore option is enabled.

[3]**txBackoff:** This parameter is the common output power backoff (in dB scale) for all TX antennas. The backoff value in this field defines how much the transmit power for each transmit antenna should be reduced from the maximum in dB scale. It is important to note that the typical output power of the xWRL6432/1432 device per TX is around 11dBm. Hence, 0dB backoff corresponds to the maximum power level available. For xWRL1432, TX power backoff settings more than 13 dB have not been characterized for RF performance. Safety diagnostic monitors are not supported for these backoff settings. TX backoff parameter should be configured according to the detection range requirements of the use case and transmission requirements mandated by the regulations, which may differ in different regions of the world.

[4]**flashOffset:** Address offset in the flash to be used while saving or restoring calibration data. Make sure the address doesn't overlap the location in FLASH where application images are stored and has enough space for saving factory calibration data. This field is don't care if both save and restore are disabled. The

Flash address range is 0x0 to 0x1FFFFF (16Mb). Actual Factory calibration data is of size 128 bytes. It is recommended to use the last sector of flash memory starting with address 0x1FF000.

3.2 Detection Layer Parameters

In the following subsections, the detection layer configuration parameters summarized in Table 2 and Table 3 are detailed. Similar to the previous section, each table presents the data type and description of each parameter along with the suggested values in an example scenario (the minor motion detection use case up to 10m with a low-bandwidth option). Recall that the data types are only classified as 'int' for integer numbers and 'float' for floating-point numbers. This document does not cover the actual byte size of the parameters (uint8, int32, etc.). Instead, the supported values or valid ranges in each parameter are given in the corresponding section. For more details about the actual data size, please refer to the SDK [1].

3.2.1 Processing Chain Configuration

The following command configures the parameters of the detection layer processing chain, which is detailed in Section 2.

sigProcChainCfg 64 2 2 0 4 4 0 20			
Parameter	**Type**	**Value**	**Description**
azimuthFftSize	int	64	Azimuth FFT size. Suggested to set as power of 2. Refer to the note below[1] for the details.
elevationFftSize	int	2	Elevation FFT size. Suggested to set as power of 2. Refer to the note below[1] for the details.
motDetMode	int	2	Major, minor, or auto mode selection flag: 1: Major motion detection only. 2: Minor motion detection only. 3: Auto detection mode. Refer to the note below[2] for the details.
coherentDoppler	int	0	Coherent/non-coherent Doppler selection flag: 0: Non-coherent integration along Doppler dimension. 1: Select maximum (coherent) peak in Doppler dimension. 2: Non-coherent integration along Doppler dimension to create the detection matrix (range-angle heatmap), and find maximum peak index to estimate the Doppler in the point cloud. Refer to the note below[3] for the details.

numFrmPerMinorMotProc	int	4	Number of frames included for minor motion detection. Refer to the note below[4] for the details.
numMinorMotionChirpsPerFrame	int	4	Number of chirp pairs (for both TXs) per frame used for minor motion detection. Refer to the note below[4] for the details.
forceMinorMotionVelocityToZero	int	0	Force Doppler value of detected points to zero in minor motion detection mode. 0: Disabled 1: Enabled This logic is added for the tracker layer processing. Refer to Section 2.7 for details behind the tracker integration.
minorMotionVelocityInclusionThr	float	20	Minor motion detected points with absolute velocity greater than this threshold (unit is m/sec) are not included in the point cloud list. This logic is added for the tracker layer processing. Refer to Section 2.7 for details behind the tracker integration.

[1]**azimuthFftSize and elevationFftSize:** These parameters define the azimuth and elevation FFT sizes in the angle estimation steps detailed in Section 2.4.3. Both parameters, which affect the search granularity (i.e., the inter-bin resolution) in the angle domain, have a significant impact on the processing load (also affects the power consumption). Hence, both parameters should be carefully optimized for the desired use case. Depending on the use case restrictions (power consumption, frame rate, etc.), small azimuth and elevation FFT sizes may be required. This will create coarse angle steps in the azimuth-elevation grid. The azimuth interpolation block (see Section 2.5.4 for details) will be used to improve the azimuth estimation accuracy.

[2]**motDetMode:** This is the critical parameter that defines the signal processing chain. As discussed in Section 2.1, the motion/presence detection algorithm will support three modes (auto, major-only, and minor-only), which can be configured by this parameter, depending on the use case requirements. If major-only or minor-only processing modes are selected, only a single point cloud set is provided to the high-level processing block. In auto mode, both detection modes are executed in the detection layer, and the generated point cloud sets from each mode are merged with proper labels to be used in the high-level processing block.

[3]**coherentDoppler:** The current implementation supports both a single peak search or non-coherent summation in the Doppler domain, as discussed in Section 2.4.4. When a single peak search mode is selected (i.e., mode 1), the strongest (i.e., maximum) peak in the Doppler spectrum is used when projecting the range-angle-Doppler data into the range-angle heatmap. In this mode, the peak index will be used as the Doppler estimation in the point cloud.

When the non-coherent integration mode is selected (i.e., mode 0), a non-coherent integration along the Doppler dimension is performed when projecting the range-angle-Doppler data into the range-angle

heatmap. In this mode, the Doppler estimation step is ignored, and the Doppler is set to zero by default in the point cloud.

When mode 2 is selected, the non-coherent integration along the Doppler dimension is used to create the range-angle heatmap (similar to mode 0), and the maximum peak index is used to estimate the Doppler in the point cloud (similar to mode 1). This mode is suggested to be used when Doppler estimation is needed (e.g., for the tracker, etc.) and the non-coherent summation is desired when generating the range-angle heatmap.

[4]**numFrmPerMinorMotProc and numMinorMotionChirpsPerFrame:** As discussed in Section 2.1, the detection layer chain utilizes the combined subsampled chirp blocks from the current and previous frames to increase the chirping window for minor and very fine motions. The parameter numFrmPerMinorMotProc defines the number of frames included for minor motion detection mode. The parameter numMinorMotionChirpsPerFrame defines how many chirp pairs (for both TXs) per frame will be used to generate the total chirp block across frames.

These parameters are closely related to the number of chirps available in a single frame and the total retention time desired to detect minor motions. It is recommended to create around 1s total retention time to detect the fine motions on a static human body (e.g., breathing, etc.). Hence, when configuring these parameters, the frame rate should also be considered. Besides, the total number of chirps in the chirp block created across multiple frames must be a power of 2 (to be able to run the clutter removal step discussed in Section 2.4.2). When auto mode is configured in motDetMode, the total number of chirps for major and minor mode radar cubes (i.e., single and multiple frames blocks, respectively) must be the same size to run the same processing chain for both radar cubes.

3.2.2 Peak Detection Configuration

In the motion/presence detection processing chain, the 2-pass CFAR algorithm is applied directly to the generated 2D range-azimuth heatmap (as discussed in Section 2.5). In this 2-pass CFAR algorithm, the first-pass scan on each angle checks the CFAR condition across the range bins. For each detected point at (rangeIndex, angleIndex), the second pass checks the condition across the angle bins to confirm the first pass detections. In the first pass, different versions of the CFAR method are supported. For details about the HWA CFAR processing and the related parameters, refer to the HWA user guide [9]. The following table summarizes the parameters used to configure the HWA CFAR engine. For details about each parameter, refer to the subsequent notes.

cfarCfg 2 4 3 2 0 12.0 0 0.5 0 1 1 1			
Parameter	**Type**	**Value**	**Description**
averageMode	int	2	CFAR averaging mode selection: 0: CFAR-CA. 1: CFAR-CAGO. 2: CFAR-CASO. Suggested to set 2.

winLen	int	4	One-sided noise averaging window length (in samples) of range-CFAR. Suggested to set as power of 2.
guardLen	int	3	One-sided guard length (in samples) of range-CFAR.
noiseDiv	int	2	Cumulative noise sum divisor expressed as a shift. Sum of noise samples is divided by 2^noiseDiv. Should be set as log2(winLen).
cyclicMode	int	0	Cyclic mode or wrapped around mode: 0: Disabled. 1: Enabled.
thresholdScale	float	12.0	Threshold factor of range-CFAR in dB scale (20log10).
peakGroupingEn	int	0	Peak grouping (in range domain) enable/disable flag: 0: Disabled. 1: Enabled.
sideLobeThreshold	float	0.5	Sidelobe threshold (in linear scale) in azimuth domain to declare a local peak as a valid detection.
enableLocalMaxRange	int	0	Extracting only local max in range: 0: Disabled. 1: Enabled: If the detected point is not local maximum in the range domain, exclude it from the detection.
enableLocalMaxAzimuth	int	1	Extracting only local max in azimuth: 0: Disabled. 1: Enabled: If the detected point is not local maximum in the angle domain, exclude it from the detection.
interpolateRange	int	1	Interpolation in range enable/disable flag: 0: Disabled. 1: Enabled.
interpolateAzimuth	int	1	Interpolation in azimuth enable/disable flag: 0: Disabled. 1: Enabled.

The following parameter configures the HWA CFAR engine to decide the version of the CFAR-CA method performed in range domain per angle bin.

- **averageMode:** This field configures the mode of the CFAR method in any of the following options:
  - CFAR CA (cell averaging): The average of both left and right reference windows is used as noise estimation for CFAR detection.
  - CFAR CAGO (cell averaging with greater of selection): The greater average of the left and right reference windows are used as noise estimation for the CFAR detection.
  - CFAR CASO (cell averaging with smaller of selection): The smaller average of the left and right reference windows is used as noise estimation for the CFAR detection.

The following CFAR window parameters in the range domain, which are used to configure the HWA CFAR engine, define a sliding window to calculate the local noise floor to be compared with the cell under test (CUT).

- **winLen:** This field is the CFAR reference window length (in terms of the number of samples) for the range domain CFARs. The given value is empirically the best choice, and the user may want to use it as a default setting.

- **guardLen:** This field is the CFAR guard window size (in terms of the number of samples) in the range domain. When calculating the detection threshold for the CUT, the left and right guardLen of samples will be excluded from noise accumulation. For example, if we have a target with an area of $0.5m^2$ reflecting radar energy, we do not want samples in those areas being counted as noise samples to raise the detection threshold. This way, there will be a richer point cloud detection out of the CFAR step. The user should adjust the setting based on the chirp configuration and derived inter-bin resolution, as well as the typical target size within the scene. These settings have already been adjusted for the desired effect on people as intended targets with the provided chirp configuration. It is also important to make sure that 2×(winLen+guardLen) < numRangeBins condition is satisfied.

The following parameters configure the implementation in HWA CFAR engine.

- **noiseDiv:** This parameter specifies the division factor with which the noise sum calculated from the left and right noise windows are divided in order to get the final surrounding noise average value. The division factor is equal to 2^noiseDiv. Therefore, only powers-of-2 division is possible, even though the number of samples specified in winLen is not restricted to powers of 2. The surrounding noise average value obtained after the division is multiplied with thresholdScale (in linear scale) to determine the final threshold used to compare the cell under test for detection.

- **cyclicMode:** This parameter specifies whether the CFAR-CA detector needs to work in cyclic mode or in non-cyclic mode. When this field is 0, the CFAR detector works in non-cyclic mode, and when it is 1, it works in cyclic mode. These two modes are different in the way the samples at the edges are handled. In cyclic mode, the CFAR-CA detector needs to wrap around the edges in a circular manner.

- **peakGroupingEn:** This parameter specifies whether peak grouping should be enabled. When this field is 0, peak grouping is disabled, which means that a peak is declared as detected as long as the cell under test exceeds the threshold. On the other hand, if this field is 1, then a peak is declared as detected only if the cell under test exceeds the threshold, as well as, if the cell under test exceeds the two neighboring cells to its immediate left and right (local maximum).

The following CFAR threshold parameter is defined as the ratio of the local noise floor to the CUT, above which a detected point is declared. The lower the thresholds, the more false detections will occur on random noise, object sidelobes, and other spurious environmental effects. The higher the thresholds, the fewer points from the objects of interest are detected.

- **thresholdScale:** This is the relative threshold for the first-pass (in the range domain) CFAR. After noise power (noisePower) is calculated based on the winLen and guardLen, the final detection threshold will be calculated by (noisePower × thresholdScale), where noisePower is the estimated

noise power in the reference windows. If the power of CUT is greater than the threshold, it will be declared as a detected point. Note that the scale in this parameter is configured in dB scale (20log10).

This threshold setting impacts detection performance significantly. The lower the threshold, the lower confident detections may show up. The user would see more noisy points around the target and more multipath ghost targets forming between the real target and strong reflectors such as metal beams and walls. On the other hand, if the threshold is too high, the user will lose a detected point that potentially carries valuable information about the target. Loss of this information would affect the performance of the final motion/presence decision in the ROI, which rely heavily on the statistics/distribution of the range/angle/SNR information of detected points from a target. The suggested values are the best empirical numbers TI obtained, but the user might need to adjust them for their customized setup.

The following parameter configures the second-pass search step in the angle domain to confirm the range-domain detected points. The details of the second-pass confirmation step are discussed in Sections 2.5.2 and 2.5.3

- **enableLocalMaxRange:** This parameter is used in combining with the second-pass search. When this parameter is enabled, the algorithm will check whether the detected point is a local maximum in the range domain or not. When this field is enabled, if the detected point is not a local peak in the range domain, the peak is excluded from the detection list.

- **enableLocalMaxAzimuth**: This parameter is used in combining with the second-pass search. When this parameter is enabled, the algorithm will check whether the detected point is a local maximum in the angle domain or not. When this field is enabled, if the detected point is not a local peak in the azimuth domain, the peak is excluded from the detection list.

- **sidelobeThreshold:** This parameter is used in combining with the localMaxAzimuthDomain field. In the second-pass, if the detected range-point is a local maximum in the angle domain and if its power exceeds the (sidelobeThreshold × peakPower), where peakPower is the power of the strongest peak in the same range bin, the algorithm will confirm it as a detected range-azimuth point. If the detected point peak is less than the (sidelobeThreshold × peakPower) along the azimuth direction, the peak is excluded from the detection list. This step is to confirm that the detected peak is not a sidelobe of the maximum peak.

The following parameters enable/disable the range and angle interpolation steps in the peak detection processing.

- **interpolateRange and interpolateAzimuth:** As discussed in Section 2.5.4., an interpolation algorithm is adopted to provide substantial refinement of the range and azimuth estimations from the CFAR detection step. These two parameters can enable/disable these processing blocks. In the motion/presence detection processing chain, it is suggested to keep both interpolation blocks enabled.

3.2.3 Region of Interest Configuration

The following command defines the sensor azimuth and elevation field-of-views (FOVs). The FOV values in this command define the total angular extent observed at both sides of the sensor. The detected points outside of this angular FOV interval are not exported in the detection list. The suggested values are tuned according to the antenna radiation patterns of the IWRL6432 EVM.

aoaFovCfg -70 70 -40 40			
Parameter	**Type**	**Value**	**Description**
minAzimuthDeg	int	-70	Minimum azimuth angle (in degrees) that specifies the start of field of view.
maxAzimuthDeg	int	70	Maximum azimuth angle (in degrees) that specifies the end of field of view.
minElevationDeg	int	-40	Minimum elevation angle (in degrees) that specifies the start of field of view.
maxElevationDeg	int	40	Maximum elevation angle (in degrees) that specifies the end of field of view.

The following command configures the sensor boundary in the range domain. The values in this command define the total extent observed by the sensor in range. The detected points outside of this interval are not exported in the detection list.

rangeSelCfg 0.1 10.0			
Parameter	**Type**	**Value**	**Description**
minMeters	float	0.1	Minimum range of exported detected points.
maxMeters	float	10.0	Maximum range of exported detected points.

minMeters: This parameter is the minimum range that will be included in the created point cloud. For the example chirp configuration, a value of 0.1m in this field implies that the detected points closer than 10cm within the sensor will be excluded from the point cloud. Discarding these range bins also reduces the throughput of the sensor. If the end application has no interest in detecting the close-by range, these bins can be discarded to save resources.

maxMeters: This parameter is the maximum range that will be included in the created point cloud. Depending on the chirp configuration and derived maximum unambiguous range, this setting dictates the range furthest away from the sensor (typically the range boundary of the scene) that will be included in the detection result. For the example chirp configuration, a value of 10m in this field implies that distances beyond 10m will not have any detection in the point cloud. Discarding these range bins also reduces the throughput of the sensor. If the end application has no interest in detecting the far range, these bins can be discarded to save resources.

3.2.4 Clutter Removal Configuration

The following parameter enable/disable the clutter removal step discussed in Section 2.4.2. Because the main goal of the motion/presence detection or the target tracking is to detect/track motions in the scene, this parameter must always be set to 1. This parameter is included in the cfg file for debugging or when analyzing the reflections from the static scenes. Please refer to Section 2.4.2 for details about the clutter removal processing.

clutterRemoval 1			
Parameter	**Type**	**Value**	**Description**
enabled	int	1	Configure static clutter removal: 0: Disabled. 1: Enabled. Always set to 1 if it is desired to detect moving objects only (person, etc.) in the scene.

3.2.5 Antenna Pattern Configuration

The motion/presence detection demo can support different antenna configurations. Although the default parameters of the demo are configurated according to the xWRLx432 EVM, the user can change the layout using the following command. In this command, the row and column index of each antenna define the virtual antennas' physical location index (0, 1, 2, …) in the elevation and azimuth domains, respectively, as illustrated in Figure 32. In other words, for the xWRLx432 EVM antenna pattern, these index values indicate two elevation rows (specified by 0 and 1), and four azimuth columns (specified by 0, 1, 2, 3) on each elevation row, as shown in Figure 32.

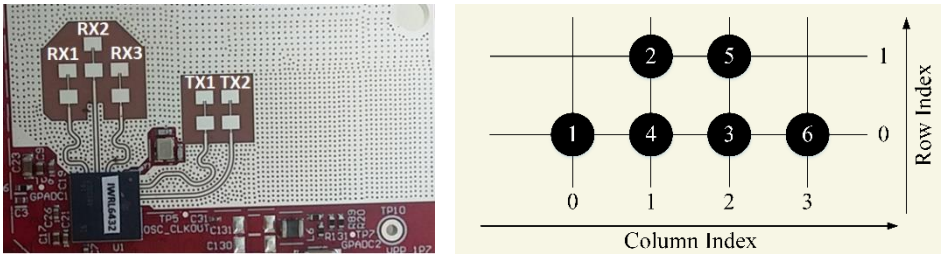


Figure 32: Virtual antenna pattern and the corresponding antenna index for xWRLx432 EVM.

antGeometryCfg 0 0 1 1 0 2 0 1 1 2 0 3 2.418 2.418			
Parameter	**Type**	**Value**	**Description**
ant1IdxRow	int	0	Row index of virtual antenna 1 (TxAnt1->RxAnt1).
ant1IdxCol	int	0	Column index of virtual antenna 1 (TxAnt1->RxAnt1).
ant2IdxRow	int	1	Row index of virtual antenna 2 (TxAnt1->RxAnt2).
ant2IdxCol	int	1	Column index of virtual antenna 2 (TxAnt1->RxAnt2).
ant3IdxRow	int	0	Row index of virtual antenna 3 (TxAnt1->RxAnt3).
ant3IdxCol	int	2	Column index of virtual antenna 3 (TxAnt1->RxAnt3).
ant4IdxRow	int	0	Row index of virtual antenna 4 (TxAnt2->RxAnt1).
ant4IdxCol	int	1	Column index of virtual antenna 4 (TxAnt2->RxAnt1).

ant5IdxRow	int	1	Row index of virtual antenna 5 (TxAnt2->RxAnt2).
ant5IdxCol	int	2	Column index of virtual antenna 5 (TxAnt2->RxAnt2).
ant6IdxRow	int	0	Row index of virtual antenna 6 (TxAnt2->RxAnt3).
ant6IdxCol	int	3	Column index of virtual antenna 6 (TxAnt2->RxAnt3).
antDistX	float	2.418	Antenna spacing in X dimension (i.e., between azimuth columns) in mm. This is an optional argument. If omitted, it is assumed that d = λ/2, where λ (wavelength) is computed based on the center frequency of the chirp configuration. It is assumed that the azimuth columns are uniformly distributed.
antDistZ	float	2.418	Antenna spacing in Z dimension (i.e., between elevation rows) in mm. This is an optional argument. If omitted, it is assumed that d = λ/2, where λ (wavelength) is computed based on the center frequency of the chirp configuration. It is assumed that the elevation rows are uniformly distributed.

3.2.6 Range Bias and Phase Compensation Configuration

This set of range and phase compensation parameters can be derived using a corner reflector at sensor boresight (at about a few meters). If the user decides not to compensate for the bias, the default values in the following table can be set in the configuration file.

compRangeBiasAndRxChanPhase 0.0 1 0 -1 0 1 0 -1 0 1 0 -1 0			
Parameter	**Type**	**Value**	**Description**
rangeBias	float	0.0	Range bias common for all antennas. Not used in the current SDK.
ant1PhaseReal	float	1	Phase compensation factor (real) of the 1st virtual antenna.
ant1PhaseImag	float	0	Phase compensation factor (imaginary) of the 1st virtual antenna.
ant2PhaseReal	float	-1	Phase compensation factor (real) of the 2nd virtual antenna.
ant2PhaseImag	float	0	Phase compensation factor (imaginary) of the 2nd virtual antenna.
ant3PhaseReal	float	1	Phase compensation factor (real) of the 3rd virtual antenna.
ant3PhaseImag	float	0	Phase compensation factor (imaginary) of the 3rd virtual antenna.
ant4PhaseReal	float	-1	Phase compensation factor (real) of the 4th virtual antenna.
ant4PhaseImag	float	0	Phase compensation factor (imaginary) of the 4th virtual antenna.
ant5PhaseReal	float	1	Phase compensation factor (real) of the 5th virtual antenna.
ant5PhaseImag	float	0	Phase compensation factor (imaginary) of the 5th virtual antenna.
ant6PhaseReal	float	-1	Phase compensation factor (real) of the 6th virtual antenna.
ant6PhaseImag	float	0	Phase compensation factor (imaginary) of the 6th virtual antenna.

It is also important to note that if the sensor is not calibrated and the default calibration coefficients are used in the cfg file (i.e., all 1s in real and all 0s in imaginary parts by default), the phase rotations at each transceiver should also be considered. The antenna elements of the xWRLx432 EVM are all fed from the same direction. Hence, the phase rotation for each virtual element is 1 by default. However, in the RF front-end of the xWRLx432 sensor, some virtual elements have a phase rotation of 180 degrees (i.e., -1). As seen in the table above, the corresponding phase compensation factors are multiplied with -1 if there is a phase rotation in the corresponding antenna element.

It is recommended to calibrate the sensor using a corner reflector and enabling the following command and replace these parameters with the board-specific calibration coefficients.

The procedure is implemented by the function mmwDemo_rangeBiasRxChPhaseMeasure(). It is called in the processing chain after the range processing DPU. The input to the procedure is a radar cube. The measurement procedure first calculates the range profile as the sum of the magnitude squares across all antennas on the range FFTs. Then it searches for the peak in the zone $X - D/2 \leq range \leq X + D/2$ where D = <searchWindow> and X =<targetDistance>.

The procedure then estimates the peak position using the three point parabolic interpolation and the range bias as the difference between the peak position and the configured target distance X.

The Rx channel compensation coefficients are calculated according to following equations.

From the radar cube stored as a complex16 3D array $X[chirp][antenna][range]$, the received symbols are extracted at the range index i_{peak} of the peak position corresponding to the target as

$$x(i) = X(0, i, i_{peak}) \quad i = 0, \dots, N_{antennas} - 1$$

The coefficients are calculated as

$$c(i) = \frac{x^*(i)}{|x(i)|^2} x_{min}$$

where

$$x_{min} = \min_i \{|x_i|\}$$

The coefficients are sent per frame to the host within the TLV packet.

Measurement Procedure Steps:

1. Set a strong target like corner reflector at boresight at desired distance and measure precisely the distance from the sensor.
2. Create the CLI configuration file for the measurement procedure with
    a. the <enabled> flag to 1 in CLI command measureRangeBiasAndRxChanPhase,
    b. all antennas enabled,
    c. TDM MIMO mode selected,
    d. major mode detection enabled.
3. Run the target demo. The target sends TLV with the calculated coefficients.
4. Copy one snapshot of the coefficients to clipboard.
5. Paste the copied line into the final CLI configuration file.

Note that when the sensors are calibrated, the phase rotations mentioned will already be compensated in the provided calibration coefficients. Note that the following command is not mandatory for the motion/presence detection demo. If this command is not included in the configuration file, the range bias and phase compensation measurements will be disabled by default.

measureRangeBiasAndRxChanPhase 1 2 0.2			
Parameter	**Type**	**Value**	**Description**
enabled	int	1	Enable measurement of the range bias and RX channel gain and phase imperfections: 0: Disabled. 1: Enabled.
targetDistance	float	2	Distance in meters where the corner reflector is located to be used as test object for measurement. This field is only used when the previous measurement mode is enabled. Note: the corner reflector should be located at the boresight of the sensor when measuring the phase compensation factors.
searchWin	float	0.2	Distance in meters of the search window around targetDistance where the peak will be searched. This field is only used when the previous measurement mode is enabled.

3.2.7 GUI Monitoring Configuration

The following command configures the output data of the motion/presence detection demo streamed over the UART port. Please refer to the demo implementation guide [1] for the details about the UART packet formats of each data output option.

guiMonitor 2 2 0 0 0 1 1 0 0 0 0			
Parameter	**Type**	**Value**	**Description**
pointCloud	int	2	Point cloud data transmission enable/disable flag: 0: Disable. 1: Enable point cloud in floating point format, plus side information. 2: Enable point cloud in compressed format (fixed point) Refer to the note below[1] for the details.
rangeProfile	int	2	Range profile data transmission enable/disable flag: 0: Disable. 1: Enable range profile from major mode detection. 2: Enable range profile from minor mode detection. 3: Enable both range profiles. Refer to the note below[2] for the details.
noiseProfile	int	0	Not used in the current SDK, always set to 0.
rangeAzimuthHeatMap	int	0	Range-azimuth heatmap data transmission enable/disable flag: 0: Disable.

			1: Enable heatmap from major mode detection. 2: Enable heatmap from minor mode detection. 3: Enable both heatmaps. Refer to the note below[3] for the details.
rangeDopplerHeatMap	int	0	Not used in the current SDK, always set to 0.
statsInfo	int	1	Statistics information (temperature and processing time) transmission enable/disable flag: 0: Disable. 1: Enable. Refer to the note below[4] for the details.
presenceInfo	int	1	Presence detection data transmission enable/disable flag: 0: Disable. 1: Enable. Refer to the note below[5] for the details.
adcSamples	int	0	Enable/disable the transmission of the raw ADC samples of the last two chirps of the frame: 0: Disable. 1: Enable. Refer to the note below[6] for the details.
trackerInfo	int	0	Group tracker data transmission enable/disable flag: 0: Disable. 1: Enable. Refer to the note below[7] for the details.
microDopplerInfo	int	0	Micro-Doppler (µ-Doppler) data transmission enable/disable flag: 0: Disable. 1: Enable. Refer to the note below[8] for the details.
classifierInfo	int	0	Classifier data transmission enable/disable flag: 0: Disable. 1: Enable. Refer to the note below[9] for the details.
quickEvalInfo	int	0	Enable/Disable the quick eval plots. The quick eval plots the presence info and is not useful without it being enabled 0 - Disabled 1 - Enabled

[1]**pointCloud:** This parameter enables or disables the export of point cloud (x, y, z, Doppler) and point cloud side info (SNR). Depending the motion/presence detection mode configured (auto, major-only or minor-only, as discussed in Section 2.1 and configured by motDetMode), point cloud will be the combined version of both major and minor motion detection chains.

[2]**rangeProfile:** This parameter enables/disables the export of range profiles (at the azimuth peaks of the range-azimuth heatmap generated in Section 2.4) of major and minor motion detection chains (or both in auto mode). Enabling these outputs has an impact on the throughput. Hence, these parameters should

be enabled only in the debug mode with longer frame periods (i.e., slower frame rates). It is important to configure this parameter according to the selected mode (configured by motDetMode).

[3]**rangeAzimuthHeat:** This parameter enables/disables the export of range-azimuth heatmaps (generated in Section 2.4) of major and minor motion detection chains (or both in auto mode). Enabling these outputs has an impact on the throughput. Hence, these parameters should be enabled only in the debug mode with longer frame periods (i.e., slower frame rates). It is important to configure this parameter according to the selected mode (configured by motDetMode).

[4]**statsInfo:** This parameter enables or disables the export of statistical information about the implementation, such as processing time and the UART transmission time. Both information can be used for benchmarking and performance tuning. This parameter also enables or disables the export of sensor temperature information.

[5]**presenceInfo:** When this flag is enabled, the device will send the motion/presence detection results to the host over the UART interface. As discussed in Section 2.6, the processing chain uses a cuboid-based zone mapping approach when running the high-level motion/presence detection logic. If the motion/presence decision is enabled (sending the commands discussed in Section 3.4), enabling this flag will send the binary motion/presence decision (i.e., 0 or 1) for each zone to the host. Note that if the motion/presence decision is disabled (i.e., the commands in in Section 3.4 are not provided in the configuration files), this flag will not be used.

[6]**adcSamples:** The motion/presence detection demo has the option of streaming the data in the ADC buffers (both ping and pong buffers) of the sensor RF front-end. Since the ADC buffers are overwritten at each chirping slot, this flag will only enable streaming the last chirp pairs (from ping and pong) of each frame available in this buffer. This flag is added mainly for debugging. If the ADC raw data is desired to be captured over UART using this flag, then only two chirps (i.e., one chirp per TX) per frame should be configured to get all the available chirps at every frame. If the user wants to enable more than two chirps per frame in the ADC raw data capturing mode, then it is suggested to use high-speed data capture alternatives using DCA1000 EVM [11].

[7]**trackerInfo:** When this flag is enabled, the device will send the tracking layer results to the host over the UART interface. The details of the tracking layer are given in Section 2.7. It is important to note that the tracking mode should be enabled first by sending the commands discussed in Section 3.5. Otherwise, this flag will have no impact on the processing chain. When both the tracker layer and this flag are enabled, the device will send the tracking information (position, velocity, acceleration, track-point cloud association, and other Kalman state information). The entire list of the data provided can be used in the implementation guide of the SDK [1] and the tracker documentation [2].

[8]**microDopplerInfo:** As discussed in Section 2.8, the motion/presence detection demo can extract the µ-Doppler information of each tracked object, if this processing option is enabled through the commands summarized in Section 3.6. Enabling this flag will then stream this data over the UART per tracked object.

The µ-Doppler generation block needs the tracker layer to be enabled. Hence, if the tracker layer or µ-Doppler generation block is not enabled, this flag will have no impact on the processing chain.

[9]**classifierInfo:** As discussed in Section 2.8, the motion/presence detection demo can classify tracked target objects using the feature set extracted from the µ-Doppler information. When this flag is enabled, the demo will provide the classification probabilities of each class (human or non-human). As known, a typical deep-learning based classifier model incorporates a softmax layer (as in Figure 31) to compute the probability distribution of possible outcomes (which is two in our demo). Hence, enabling this flag will then stream these probability distributions data over the UART per tracked object. Note that the probability of each class will be between [0 1], where the sum of these probabilities will be 1. This information is then used by the GUI-based application to be converted to binary tags. Similar to the previous flag, if any of the tracker layer, µ-Doppler generation or classifier blocks is disabled, this flag will have no impact on the processing chain.

[10]**quickEvalInfo:** This parameter enables/disables the quick eval plots. In the SDK visualizer, quick eval plots the presence info within 1x1 sq mtr box infront of the radar and is not useful without it being enabled. As part of Quickeval TLV, If enabled, sceneryParams (sensor position, sensorOrientation, boundary of interest) are sent to the host.

3.3 Debug Related Parameters

The following parameter is used for debugging the target code using captured raw ADC data. When this debug option is enabled, a recorded bin file path can be provided in this command. The generated point cloud by the radar device can then be analyzed in debug mode. Please refer to the SDK [1] for details about the available debug modes.

adcDataSource 0 adc_data_name.bin			
Parameter	**Type**	**Value**	**Description**
enabled	int	0	Enable/disable the target code verification using a captured raw ADC data: 0: Disable 1: Enable
fileName	string	adc_data_name.bin	Full file name including extension and data path. This is the binary file with ADC samples data to be used in debugging.

The following parameter is used to captured raw ADC data using DCA1000 EVM [11].

adcLogging 0			
Parameter	**Type**	**Value**	**Description**
enabled	int	0	ADC data logging enable: 0 - Disable. 1: Enable via DCA.

			2: Enable via SPI. NOTE: When ADC logging via DCA is enabled number of adc samples must be a multiple of 4. And the configs listed below are not applicable when SPI logging is used.
sideBandEnable	int	0	Sideband data logging:(optional argument) 0: Disable. 1: Enable. NOTE: FrameLivMonEn in sensorstart must be set to a value 2(RX_SATURATION_LIVE_MON must be enabled) for sideband data logging. And by default sideband data is disabled. (Refer to the ICD for more details about this feature)
swizzlingMode	int	2	RDIF Data Swizzling Mode enable control:(optional argument) 0: Pin0-bit0-Cycle1 Mode. 1: Pin3-bit0-Cycle1 Mode. 2: Pin0-bit0-Cycle3 Mode. 3: Pin3-bit0-Cycle3 Mode. (Refer to the ICD for more details about this feature)
scramblerMode	int	0	RDIF Scrambler Mode enable control:(optional argument) 0: Disable. 1: Enable. (Refer to the ICD for more details about this feature)
laneRate	int	0	Enable RDIF lane rate update:(optional argument) 0: Combined Lane Rate of 400Mbps. 1: Combined Lane Rate of 320Mbps. 2: Combined Lane Rate of 200Mbps. 3: Combined Lane Rate of 160Mbps. 4: Combined Lane Rate of 100Mbps. (Refer to the ICD for more details about this feature)

3.4 Motion/Presence Detection Layer Parameters

The configuration parameters presented in this section are used to configure the high-level processing layer for motion/presence decision discussed in Section 2.6. These parameters should be adjusted to match the use cases based on the particular scenery and target characteristics. An example of high-level processing parameters is listed in Table 2 (Part 4). In the following subsections, we provide detailed information on each of the CLI commands.

3.4.1 Scene Configuration

This set of parameters allows the user to configure the dimensions of the physical space in which the motion/presence detection chain will operate. These also specify the radar sensor orientation and position. It's convenient to first define mathematical spaces in which the measured data is obtained, the motion/presence detector operates, and the output is visualized. We define the following spaces which are applicable in any sensor mount configurations:

- World Space W: $\{X_w, Y_w, Z_w\}$ is Cartesian with origin at floor level of a real-world scene

- Sensor Space T: $\{X_t, Y_t, Z_t\}$ is Cartesian with origin at the sensor
- Point Cloud Space P: $\{r, \varphi, \theta, \dot{r}\}$ } is Spherical, where the center of the virtual antenna array of the radar sensor is considered to be the origin. r is the radial distance, φ is the azimuth angle, θ the elevation angle and $\dot{r}$ is the radial velocity of a measured point with respect to the sensor axis.

The sensor mounting geometry and its relation to the different measurement spaces are defined as:

- Sensor location $S_w = \{x_o, y_o, z_o\}$ is specified in the world Cartesian space where x_o, y_o, and z_o are the offsets of the sensor referenced to the origin (O).
- All configuration boxes defined by the user are also in the world Cartesian space. The boundary box is specified by the coordinate values $\{x_1, x_2, y_1, y_2, z_1, z_2\}$.

The following boundary box command allows the user to configure the physical dimensions of the space in which the motion/presence decision will be provided. For example, a boundary box can be configured to ignore the targets in the hallway outside the room or to ignore potential ghost targets that appear behind the room walls due to multipath reflections.

As discussed in Section 2.6, the motion/presence decision logic can be run in multiple zones within the entire scene. Hence, multiple boundary boxes can be configured. Therefore, the entire scene can be divided into multiple boundary boxes, and each boundary box can be sent to the device separately. This scheme will then allow running the motion/presence decision state machine per zone. Note that each box should be defined in meters in the world coordinates in Cartesian space.

mpdBoundaryBox 1 -5 5 0 10 0 3			
Parameter	**Type**	**Value**	**Description**
index	int	1	Index of the boundary box. It should start from 1 and increase sequentially if multiple boundary boxes are defined.
xMin	float	-5	Minimum horizontal distance (in meters) with respect to the origin in the world coordinates.
xMax	float	5	Maximum horizontal distance (in meters) with respect to the origin in the world coordinates.
yMin	float	0	Minimum vertical (i.e., depth) distance (in meters) with respect to the origin in the world coordinates.
yMax	float	10	Maximum vertical (i.e., depth) distance (in meters) with respect to the origin in the world coordinates.
zMin	float	0	Minimum height (in meters) with respect to the origin in the world coordinates. Note that z = 0 corresponds to the ground plane (In the current demo implementation this parameter is not taken into consideration).
zMax	float	3	Maximum height (in meters) with respect to the origin in the world coordinates (In the current demo implementation this parameter is not taken into consideration).

NOTE: In the current MPD DPU implementation, only x and y co-ordinates are taken into consideration.

The following command can be used to specify the radar sensor orientation and position. The sensor location S_w = {xOffset, yOffset, zOffset} is specified in the world Cartesian space referenced to the boundary box origin.

sensorPosition 0 0 2 0 0			
Parameter	**Type**	**Value**	**Description**
xOffset	float	0	Offset of the radar sensor position in x-axis referenced to the boundary box origin.
yOffset	float	0	Offset of the radar sensor position in y-axis referenced to the boundary box origin.
zOffset	float	2	Height of the radar sensor above the ground plane.
azimuthTilt	float	0	The azimuth tilt (in degrees) of the sensor about the axis Z_w in Figure 5b. A positive value indicates clockwise rotation.
elevationTilt	float	0	The elevation tilt of the sensor about the axis X_w in Figure 5b. A negative value indicates clockwise rotation (i.e., the tilt is towards the ground).

3.4.2 Clustering Configuration

The following command is used to cluster all the detected points and determine if that cluster qualifies as a person/target. The clustering parameters determine if a candidate point can be clustered into a clustered set. To join the set, each point needs to be within maxDistance from the clustered sets centroid. Moreover, once the set is formed, it has to have more than minPoints members. This clustering logic only runs on x and y co-ordinates. For details about the clustering logic, refer to Section 2.6.

clusterCfg 1 0.5 2			
Parameter	**Type**	**Value**	**Description**
enabled	int	1	Configure clustering logic. 0: Disable. 1: Enable.
maxDistance	float	0.5	The radius (in meters) of the neighborhood around a point (i.e., epsilon in DBSCAN algorithm). Note that the distance measure only takes the geometric difference between the point and centroid into account.
minPoints	int	2	Minimum number of neighbor points required within the epsilon radius around point.

3.4.3 State Transition Configuration

Any zone configured in Section 3.4.1 can be in one of three states i.e., empty, major, or minor. The transition from one state to another is determined by various parameters, which are configured by the following command. For details about the state transition logic, refer to Section 2.6.

NOTE: pointThre1, pointThre2, and snrThre2 are checked when the state goes from empty to minor/major or minor to major. The pointHistThre1, pointHistThre2, and snrHistThre2 are checked when the state goes from major to minor/empty or minor to empty.

majorStateCfg 4 2 20 8 6 40 4 4			
Parameter	**Type**	**Value**	**Description**
pointThre1	int	4	Number of detected points (in a single frame) needed in a zone to enter the motion/presence state. If the number of points exceeds this threshold, there is no need to check the SNR (this field is only checked when the state goes from empty to major or minor to major).
pointThre2	int	2	Number of detected points (in a single frame) needed in a zone to enter the motion/presence state. If the number of points exceeds this threshold, the snrThre2 criteria is checked (this field is only checked when the state goes from empty to major or minor to major).
snrThre2	float	20	Minimum total SNR (linear) of detected points (in a single frame) in a zone to enter the motion/presence state if the pointThre2 criteria is also satisfied.
pointHistThre1	int	8	Number of detected points (in a frame history buffer) needed in a zone to stay in the motion/presence state. If the number of points exceeds this threshold, there is no need to check the SNR (this field is only checked when the state goes from major to minor/empty).
pointHistThre2	int	6	Number of detected points (in a frame history buffer) needed in a zone to stay in the motion/presence state. If the number of points exceeds this threshold, the snrHistThre2 criteria is checked (this field is only checked when the state goes from major to minor/empty).
snrHistThre2	float	40	Minimum total SNR (linear) of detected points (in a frame history buffer) in a zone to enter the motion/presence state if the pointHistThre2 criteria is also satisfied.
histBufferSize	int	4	Size of the frame history buffer size (in frames) used in pointHistThre1, pointHistThre2, and snrHistThre2 parameters.
major2minorThre	int	4	A motion status is preserved if it recorded at least one motion detection in the last major2minorThre frames.

minorStateCfg 4 3 10 8 6 20 8 20			
Parameter	**Type**	**Value**	**Description**
pointThre1	int	4	Number of detected points (in a single frame) needed in a zone to enter the motion/presence state. If the number of points exceeds this threshold, there is no need to check the SNR (this field is only checked when the state goes from empty to minor).

pointThre2	int	3	Number of detected points (in a single frame) needed in a zone to enter the motion/presence state. If the number of points exceeds this threshold, the snrThre2 criteria is checked (this field is only checked when the state goes from empty to minor).
snrThre2	float	10	Minimum total SNR (linear) of detected points (in a single frame) in a zone to enter the motion/presence state if the pointThre2 criteria is also satisfied.
pointHistThre1	int	8	Number of detected points (in a frame history buffer) needed in a zone to stay in the motion/presence state. If the number of points exceeds this threshold, there is no need to check the SNR (this field is only checked when the state goes from minor to empty).
pointHistThre2	int	6	Number of detected points (in a frame history buffer) needed in a zone to stay in the motion/presence state. If the number of points exceeds this threshold, the snrHistThre2 criteria is checked (this field is only checked when the state goes from minor to empty).
snrHistThre2	float	20	Minimum total SNR (linear) of detected points (in a frame history buffer) in a zone to enter the motion/presence state if the pointHistThre2 criteria is also satisfied.
histBufferSize	int	8	Size of the frame history buffer size (in frames) used in pointHistThre1, pointHistThre2, and snrHistThre2 parameters.
minor2emptyThre	int	20	A motion status is preserved if it recorded at least one motion detection in the last minor2emptyThre frames.

3.5 Tracking Layer Parameters

As discussed in Section 2.7, the motion/presence detection demo utilize the same EKF based group tracker algorithm in the IWR6843 people counting demo [2]. Hence, for detailed description of the group tracker algorithm and tracker layer parameters that can be configured through the CLI interface, please refer to the group tracker implementation [2] and tuning guides [1]. The purpose of this section is to summarize the parameters to configure the tracker layer and give the user general guidelines on understanding these parameters at very high level.

The list of the configuration parameters needed to enable the tracker layer on the motion/presence detection demo is given in Table 3 (Part 4). Each line in the configuration file represents a CLI message that configures several parameters. The given example parameters at each command should be adjusted to match the use-cases based on the particular scenery and target characteristics. The tracker layer parameters can be grouped into a few sets. A high-level description of the parameter sets and the corresponding CLI command are shown in the following table. As mentioned, a separate group tracker tuning guide [1] provides detailed information on each of the CLI commands.

No	Parameter Set	CLI Commands	Description
1	Scenery Parameters	boundaryBox staticBoundaryBox	These define the dimensions of the physical space in which the tracker will operate. These also specify the

		sensorPosition presenceBoundaryBox	radar sensor orientation and position. Any measurement points outside these boundary boxes will not be used by the tracker.
2	Gating Parameters	gatingParam	These determine the maximum volume and velocity of a tracked object and are used to associate measurement points with tracks that already exist. Points detected beyond the limits set by these parameters will not be included in the set of points that make up the tracked object.
3	Track Allocation Parameters	allocationParam	These are used to detect new tracks/people in the scene. When detected points are not associated with existing tracks, allocation parameters are used to cluster these remaining points and determine if that cluster qualifies as a person/target.
4	State Transition Parameters	stateParam	The state transition parameters determine the state of a tracking instance. Any tracking instance can be in one of three states: free, detect, or active.
5	Max Acceleration Parameters	maxAcceleration	These parameters determine the maximum acceleration in the lateral, longitudinal, and vertical directions.
6	Tracker Configuration Parameters	trackingCfg	These parameters are used to enable the Tracker Module and determine the amount of memory to allocate based on maximum number of points and tracks. It also configures the radial velocity parameters (initial velocity, velocity resolution, max velocity) and frame rate at which the tracker is to operate.

3.6 Classification Layer Parameters

The configuration parameters presented in this section are used to configure the high-level processing layer for target classification discussed in Section 2.8. These parameters should be adjusted to match the requirements of different use cases. An example of target classification layer parameters is listed in Table 3 (Part 5). In the following subsections, we provide detailed information on each of the CLI commands.

3.6.1 Micro-Doppler Generation and Feature Extraction

microDopplerCfg 1 0 0.5 0 1 1 12.5 87.5 1			
Parameter	**Type**	**Value**	**Description**
enabled	int	1	Micro-Doppler (µ-Doppler) and feature extraction processing enable/disable flag: 0: Disable. 1: Enable.

genApproach	int	0	The angle spectrum generation approach (in the azimuth domain) when creating the micro-Doppler (µ-Doppler) spectrum per track: 0: FFT. 1: Beamforming. Refer to the note below[1] for the details.
targetSize	float	0.5	The target size (in xy-domain) to be used to when extracting the µ-Doppler around the corresponding centroid. The target size (in meters) configured in this parameter should be the estimated total size of the desired target. The 0.5m is tuned to be used for a typical human target (0.5m by 0.5m target size in xy-domain). This flag is used only when the genApproach flag is selected as 0 (FFT). Refer to the note below[1] for the details.
magnitudeSquared	int	0	If this flag is enabled, the generated µ-Doppler spectrum will be magnitude squared: 0: Keep the magnitude spectrum as is. 1: Take the magnitude square (i.e., power spectrum). If the pretrained classifier model will be used, this parameter should be set to 0 since the model has been trained in magnitude spectrum (not power spectrum).
circShiftCentroid	int	1	If this flag is enabled, the generated µ-Doppler spectrum will be circularly shifted around the estimated target velocity provided by the tracker: 0: Keep the µ-Doppler spectrum as is. 1: Circularly shift the µ-Doppler spectrum around the estimated target velocity from the tracker. If the pretrained classifier model will be used, this parameter should be set to 1 since the model has been trained with the µ-Doppler spectrum shifted around the track velocity. Refer to the note below[2] for the details.
normalizedSpectrum	int	1	If this flag is enabled, the generated µ-Doppler spectrum will be normalized between [0 1]: 0: Keep the µ-Doppler spectrum as is. 1: Normalize the µ-Doppler spectrum between [0 1]. If the pretrained classifier model will be used, this parameter should be set to 1 since the model has been trained with the µ-Doppler spectrum normalized between [0 1]. Refer to the note below[3] for the details.
interceptThrLowFreq	float	12.5	The power ratio (%) used to compute the lower envelope (D_{low}) feature from the µ-Doppler spectrum. This parameter is common for all the tracked objects. Refer to the note below[4] for the details.
interceptThrUpFreq	float	87.5	The power ratio (%) used to compute the upper envelope (D_{up}) feature from the µ-Doppler spectrum. This parameter is common for all the tracked objects. Refer to the note below[4] for the details.

specShiftMode	int	1	If this flag is enabled, the generated µ-Doppler spectrum will be circularly shifted around the mean Doppler: 0: Keep the µ-Doppler spectrum as is. 1: Circularly shift the µ-Doppler spectrum around the mean Doppler. If the pretrained classifier model will be used, this parameter should be set to 1 since the model has been trained with the µ-Doppler spectrum shifted around the mean Doppler. Refer to the note below[5] for the details.

[1]**genApproach:** As detailed in Section 2.8, two approaches are adapted when creating the µ-Doppler spectrogram per track using the Doppler spectrum around its centroid: (1) extract the µ-Doppler spectrum at the target centroid only, (2) extract the µ-Doppler spectrum by configuring a fixed target size around the centroid and accumulate the Doppler spectrums within this target boundary. If the approach 2 (accumulate Doppler spectrums within a target boundary) is desired, then this genApproach flag should be set to 0 (FFT). Then the target size will be configured by the targetSize parameter. If approach 1 (create the µ-Doppler spectrum of the tracks at their centroids only) is desired, the genApproach flag should be set to 1 (Beamforming) since it will be more efficient than FFT at a single angle. If the user has any prior knowledge about the approximate target sizes, it is suggested to use approach 2 with genApproach is 0 (FFT), since the accumulation of Doppler at different points of the targets can lead more SNR and accuracy.

[2]**circShiftCentroid:** As discussed in Section 2.8, the processing chain has the option of circularly shifting the generated µ-Doppler spectrum per track around the corresponding tracker velocity. This processing step is added to reduce the dependency on numerous possible target scenarios for more robust object classification with less complexity using the pretrained 1D CNN based classifier. On the other hand, the user always has the option of disabling this flag to generate the original µ-Doppler spectrum for each track. However, it is important to remember that the pretrained classifier model will not work for this option. Hence, the user should train another model using the data captured with this option disabled.

[3]**normalizedSpectrum:** This parameter is added to normalize the generated µ-Doppler spectrum between [0 1]. This option helps to eliminate the effect of SNR in the classifier model. Similar to the other flags, the user always has the option of disabling this flag to generate the original µ-Doppler spectrum for each track. Then another classifier model should be trained using the data captured with this option disabled. When normalizing the spectrum, the simple rescaling approach is used. This approach changes the distance between the minimum and maximum values in the µ-Doppler spectrum by stretching or squeezing the points along the number line. The z-scores of the data are preserved, so the shape of the distribution remains the same. The equation used to rescale the spectrum to the interval [0 1] is $X_{rescaled} = [(X - X_{min})/(X_{max} - X_{min})]$, where X_{min} and X_{max} are the minimum and maximum values of the data X, respectively.

[4]**interceptThrLowFreq** and **interceptThrUpFreq:** It is important to note that the major features (upper and lower envelopes) detailed in Section 2.8, which are extracted using the frequency intercepts of the occupied band, are heuristically configured. A proper configuration of the spectrum intercepts affects the extracted upper and lower envelope signatures hence the overall classification accuracy. Configuring the

intercepts with inaccurate parameters may lead to either missing some weak signatures or result in a bad representation of the spectrum. In our end-to-end implementation, a significant amount of tuning effort (both in the signal processing and classification algorithm levels) is made to achieve the optimal representations of the upper and lower envelopes of the generated µ-Doppler spectrums, and the best envelope representations are achieved with power intercept ratios of [%12.5 %87.5] when the other flags (magnitudeSquared, circShiftCentroid, and normalizedSpectrum) are configured to their default values given. When the user updates any of these flags, the upper and lower envelopes should be retuned accordingly.

[5]**specShiftMode:** As discussed in Section 2.8, the processing chain has the option of circularly shifting the generated µ-Doppler spectrum per track around estimated mean Doppler. This processing step is added to eliminate the errors made by tracker when estimating the target centroid state information (both position and velocity). Similar to the other flags, the user always has the option of disabling this flag to generate the original µ-Doppler spectrum for each track. Then another classifier model should be trained using the data captured with this option disabled.

3.6.2 Target Classification

classifierCfg 1 3 4			
Parameter	**Type**	**Value**	**Description**
enabled	int	1	Classifier processing enable/disable flag: 0: Disable. 1: Enable.
minNumPntsPerTrack	int	3	Minimum number of points required for a track to run the classifier for it. Refer to the note below[1] for the details.
missTotFrmThre	int	4	The maximum number of frames allowed which does not have enough number of points for a specific track. Refer to the note below[1] for the details.

[1]**minNumPntsPerTrack** and **missTotFrmThre:** As discussed in Section 2.8, for robustness in the classification algorithm, some more pre-processing steps are implemented. In the first step, if a certain track has no associated points at some frames, we avoid updating the feature set because the extracted information will not be reliable. When such cases happen, we wait for the next frames to extract new features and delay the classification decision. The first parameter minNumPntsPerTrack controls this logic to control the minimum number of points required. Then the missing samples in the frame axis are interpolated (linear interpolation) using the neighbors. On the other hand, for robustness, we should limit the maximum number of missing frames in the complete block. The reason is, if the total number of missing samples exceed some threshold, then the generated feature set (after the interpolation) will lose it reliability. The second parameter missTotFrmThre controls this logic to control the maximum number of frames allowed which does not have enough number of points for a specific track.

3.7 Profile Switching Parameters

This feature enables seamless transitions between high-performance tracker processing and low-power presence detection through the designed state machine. It additionally achieves power saving by intelligent configuration switching, based on the detected scene. The flow diagram is shown below:

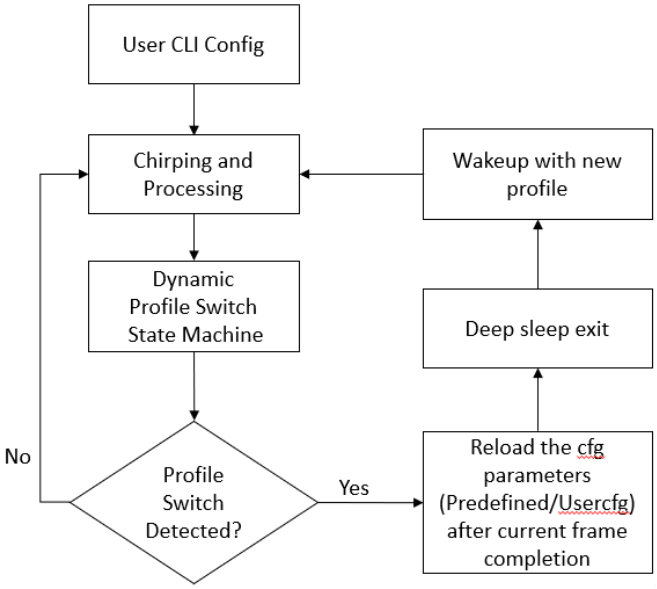


Figure 33: The flow diagram for profile switching between high-performance tracker processing and low-power presence detection.

This feature can be added to the demo code by using syscfg, as it is disabled by default. And, the configuration parameters can be populated via the CLI command - "profileSwitchCfg" which has the following parameters:

profileSwitchCfg 1 25 100			
Parameter	**Type**	**Value**	**Description**
switchCfgEnable	int	1	Profile switch enable: 0: Disable. 1: Enable. Enabling low power mode in lowPowerCfg CLI is a prerequisite to enable this feature.
frmPretoTrack	int	100	Threshold on number of frames for switching from presence to tracker state.
frmTracktoPre	int	25	Threshold on number of frames for switching from tracker to presence state.

Users can initiate the demo with either the presence/tracker configuration. Once the set threshold is hit, the sensor smoothly switches to the alternate configuration (tracker/presence) predefined in the demo application. When the sensor returns to the initial config state based on the state machine, it takes the saved user-configured parameters.

For example, if a user starts the demo with the presence configuration with profileSwitchCfg enabled, the sensor automatically transitions to the tracker configuration when continuous presence is detected for frmPretoTrack frames. If no tracker objects are detected for frmTracktoPre frames, the demo reverts to the user-provided presence configuration. This cycle repeats based on the sensing scene. In turn, power saving is also achieved by staying in the low-power presence configuration unless a constant movement is detected for the configured number of frames.

NOTE: This feature can be enabled only when the lowPowerCfg CLI command is enabled. This ensures maximum power savings possible and an optimal demo experience overall.

4 Parameter Tuning and Performance

This section guides the user to tune the configuration parameters discussed in the previous sections and presents how the motion/presence detection performance is affected when these parameters are changed. This section also describes some commonly encountered scenarios where the user may need to tune the configuration parameters discussed in Section 3 to obtain optimal performance in the processing chain.

4.1 Effect of the Radar Parameters on the Physical Requirements

First, we detail the physical parameters in real-world scenarios and their relations to the radar system parameters discussed in this document. In a real-world scenario, one of the most critical limitations that the user should consider in their applications is the sensor's ability to distinguish between targets that are very close in either range, velocity, or angle domains, which is typically known as resolution:

- **Range resolution:** The range resolution defines the minimum separation in the range domain at which the radar can distinguish two targets and detect them as separate objects.
- **Velocity resolution:** The velocity resolution is the minimum radial velocity difference between two objects for the radar to detect them as separate objects in the Doppler domain.
- **Angle resolution:** The angle (i.e., azimuth and elevation) resolution defines the minimum separation in the angular domain at which the radar can distinguish two targets and detect them as separate objects.

Besides, the following parameters are also essential to define the physical extent of the target scene:

- **Maximum unambiguous range:** This parameter specifies the maximum target range that the radar can unambiguously resolve. Targets at ranges beyond the maximum unambiguous range r_{max} cannot be detected because only the range interval $[0, r_{max}]$ is available at the range FFT output.
- **Maximum detection range based on SNR:** The maximum range at which a human can be detected. This parameter is computed by the link budget formula discussed below, which is a function of the detection SNR, radar cross-section of the object, RF performance of the radar device, antenna gains, and the chirp parameters.
- **Maximum unambiguous velocity:** This parameter specifies the maximum magnitude value of the target's radial velocity that the radar can unambiguously resolve. Targets detected at velocities whose magnitude is greater than the maximum unambiguous velocity v_{max} are wrapped into the interval $[-v_{max}\ v_{max}]$. In other words, objects moving faster than this value may have incorrect velocity measurements.

Table 5 summarizes the radar system parameters that govern the physical bounds of the processing chain presented above. Besides, it provides the corresponding values of each parameter in an example use case (the major motion detection use case up to 15m with a low-bandwidth option). To get a detailed understanding of the radar system parameters, please refer to the mmWave training series [3] and the

application report in [8]. The mmWave sensing estimator [12] is also a great tool for calculating different chirp configurations based on scene parameters.

Table 5. The sensor and physical parameters of the motion/presence detection chain for an example use case.

Parameters	Unit	Low Bandwidth (15m)
Transmit power, P_{TX}	dBm	10
Valid Bandwidth, B	MHz	486
Chirp time, T_c	µs	25.6
Chirp repetition period, T_r	µs	300
Number of chirps per frame (per TX antenna), N_c	-	32
Maximum beat frequency, $f_b = 0.9 \times f_s/2$	MHz	2.25
Center frequency, f_c	GHz	61.32
Maximum unambiguous range, $r_{max,u} = (cf_b)/(2K)$	m	17.76
Range resolution, $\delta_r = c/2B$	cm	30.84
Maximum unambiguous velocity, $v_{max} = c/(4f_cT_r)$	m/s	4.08
Velocity resolution, $\delta_v = c/(2N_cf_cT_r)$	m/s	0.25
Maximum detection range based on SNR, $r_{max,d} = \sqrt[4]{\frac{\sigma\, P_{TX}\, G_{TX}G_{RX}\, P_{MUX}\, \lambda^2\, T_cN_cN_{TX}N_{RX}}{(4\pi)^3 k\, T_e\, \eta\, L\, \mathrm{SNR}}}$	m	24.65

It is important to emphasize that the bandwidth (B) and chirp time (T_c) refer to the valid sweep bandwidth and ADC sampling time, respectively, discussed in Section 3.1.2. The chirp repetition period (T_r) is $(T_{\mathrm{idle1}} + T_{\mathrm{idle2}} + 2T_{\mathrm{ramp}})$ for two TX antennas $(N_{\mathrm{TX}} = 2)$, where T_{idle1} and T_{idle2} are the idle times per chirp (see Figure 8) configured as discussed in Sections 3.1.2 and 3.1.3, and T_{ramp} is the ramp end time of the chirp. The center frequency f_c in Table 5 is computed as

$$f_c = \underbrace{f_0 + T_{\mathrm{ADC}}K}_{\hat{f}_0} + B/2, \qquad \hat{f}_0\text{: effective start frequency} \tag{3}$$

where f_0 is the start frequency, T_{ADC} is the ADC start time, and K is the frequency slope configured as discussed in Section 3.1.2. The maximum beat frequency f_b in Table 5 depends on the ADC sampling frequency f_s discussed in Section 3.1.2. In the real sampling mode, the IF bandwidth is limited to $(0.9 \times f_s/2)$ or $(0.8 \times f_s/2)$ depending on the configured filter characteristic (using the dfeFirSel parameter discussed in Section 3.1.2).

Finally, the following are the other physical parameters used in the given analysis:

- Speed of light: $c = 3 \times 10^8\ m/s$
- Wavelength: $\lambda = c/f_c$
- Boltzmann's constant: $k = 1.38 \times 10^{-23}$
- Ambient temperature: $T_e = 293°K$

- Radar cross-section: $\sigma = 1\text{m}^2$
- Combined TX/RX antenna gain: $G_{TX}G_{RX} = 0\text{dB}$
- Multiplexing gain (BPM): $P_{MUX} = 3\text{dB}$
- Total system loss: $L = 5\text{dB}$
- Noise figure of the receiver: $\eta = 15\text{dB}$
- Required detection SNR: $\text{SNR} = 9\text{dB}$
- Total virtual antennas: $N_{TX}N_{RX} = 6$

4.2 How to Configure the Range

This section lists all the parameters that affect the range measurements and presents the relation of each parameter with the range boundary and resolution in real-world scenarios. First, the user must make sure that the chirp configuration that the device is configured to acquire data extends to the range at which we want to detect people/targets and matches with the configured mpdBoundaryBox. The following table establishes the relationship between the system's range requirement and the FMCW chirp waveform parameters discussed in Section 3.1.2.

Commands	Parameters	Tuning Guide
mpdBoundaryBox	xMin, xMax, yMin, yMax, zMin, zMax	Start by properly setting the scenery parameters. Many ghosts caused by multipath reflections from the wall will appear outside of the detection area. These can be immediately ruled out if the scenery parameters are properly set. Besides, the following chirp configurations of the device that manages the range extension should match the boundary box of the scene.
chirpTimingCfg	chirpRfFreqSlope	This parameter affects the bandwidth. Therefore, it manages both the range resolution and maximum unambiguous range. Increasing the frequency slope improves the range resolution but decreases the maximum unambiguous range. The user must ensure that the occupied bandwidth is within the sensor and region-based regulation (FCC, etc.) limits.
chirpComnCfg	numOfAdcSamples	This parameter affects the chirp time and bandwidth. When the frequency slope and the sampling rate are constant, it manages the range resolution. Increasing the number of ADC samples results in a wider bandwidth that improves the range resolution. This parameter also affects the valid chirping time and maximum detectable range based on SNR. On the other hand, this parameter affects the radar cube size and required processing power. Changing this parameter may result in the system running out of memory and MIPS.
	digOutputSampRate	This parameter affects the chirp time and bandwidth. When the frequency slope and the number of ADC samples are constant, it manages both the range resolution and

		maximum unambiguous range. In this case, increasing the sampling rate results in a narrower bandwidth that degrades the range resolution but increases the maximum unambiguous range. This parameter also affects the valid chirping time and maximum detectable range based on SNR

TI's mmWave sensors have a backoff capability to reduce the transmit power from the maximum that can be tuned according to the detection range requirements of different use cases. If the transmit backoff is reduced to 0dB (corresponds to the maximum available transmit power), the user can improve the detection range, as discussed in Section 4.1. Configuring the power as too high (i.e., using a backoff value close to 0dB) may cause stronger multipath reflections and higher false alarms. On the other hand, too low TX power will decrease the detection range performance, which may result in more missed detections.

Commands	Parameters	Tuning Guide
factoryCalibcfg	txBackoff	This parameter affects the maximum detection range and should be configured based on the use case and regulation requirements.

As discussed in Section 2.5, the 1D CFAR-CASO algorithm is applied across the range domain per angle bin for peak detection. In the CFAR-CASO scheme for a 1D data set in Figure 34, the noise power is estimated as the average power of the reference cells within a configured window around each cell under test. The CFAR-CASO detector assumes that the reference cells do not contain any signals from targets. Therefore, guard cells prevent noise estimation from signals leaking into the reference cells from the target peaks.

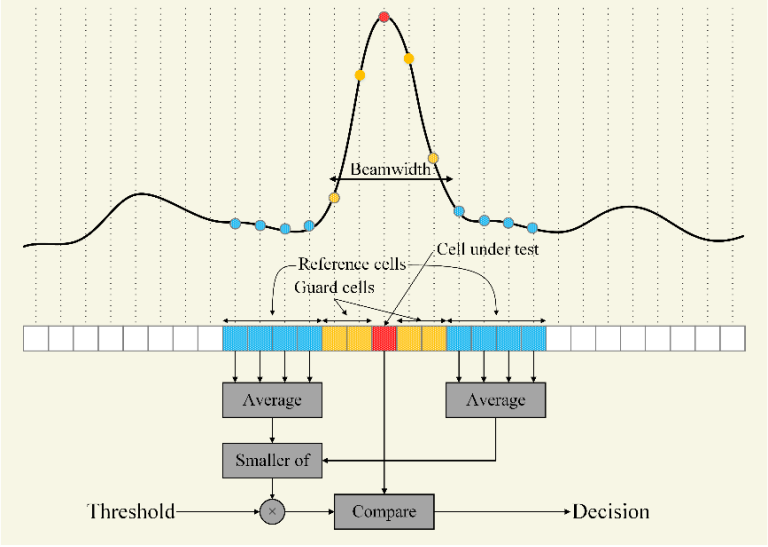


Figure 34: The 1D CFAR-CASO detector estimates the noise variance for the cell under test from the neighboring reference cells.

The reference and guard window sizes in Figure 34 significantly affect detection performance and should be carefully tuned according to the range parameters configured above. As discussed in Section 3.2.2 and illustrated in Figure 34, both window sizes are configured in terms of the number of samples. Hence, depending on the chirp configuration and derived inter-bin range resolution, the user must accordingly tune the following parameters. For example, a higher chirp bandwidth results in a sharper peak in the range FFT output, and therefore, the guard window size can be reduced to improve the noise estimation accuracy. Similarly, when the number of ADC samples or the sampling rate change, the user should adjust the reference and guard cell sizes accordingly.

Commands	Parameters	Tuning Guide
cfarCfg	winLen	These parameters are defined in the number of samples. Therefore, when the range resolution and the number of ADC samples are changed, these parameters should be adjusted accordingly.
	guardLen	

4.3 How to Configure the Motion Sensitivity

Optimizing the motion sensitivity (i.e., the velocity resolution) is a crucial task in the motion/presence detection processing chain to be able to distinguish between purely static objects (such as chairs, tables, etc.) and the people who remain stationary or with very slow motion in the scene. As discussed in Section 2, to successfully leave the signals scattered from the moving objects after the static clutter removal step, a high-velocity resolution is needed to improve the motion sensitivity. On the other hand, when the velocity resolution is increased, the processing chain becomes more sensitive to undesirable motion artifacts such as sensor vibration. The user should consider a proper tradeoff between detection and false alarm performance according to the target environment.

As presented in Section 4.1, the increased total chirp observation window consists of multiple successive chirps, and the inter-chirp time affects the maximum unambiguous velocity and velocity resolution. This section provides a tuning approach for the system's velocity requirements configuring the related radar parameters.

Commands	Parameters	Tuning Guide
frameCfg	numOfChirpsInBurst burstPeriodicity numOfBurstsInFrame	These parameters affect the total chirping window in a single frame. Increasing the number of total chirps improves the velocity resolution but also reduces the total available processing time budget within the frame period. Hence, the user must ensure that the total chirping time and the required processing time can fit into a single frame period. On the other hand, these parameters affect the radar cube size, required processing power, and average power consumption.

	framePeriodicity	In the minor motion processing mode (i.e., processing across multiple frames, as discussed in Section 2.1), the frame periodicity affects the total chirping window. Hence, increasing the inter-frame time improves the velocity resolution. On the other hand, the system performance will degrade when detecting the major motions (i.e., when too much dynamicity occurs in larger inter-frame durations).
chirpTimingCfg	chirpIdleTime	These parameters affect the inter-chirp time, hence both the velocity resolution and maximum unambiguous velocity. When the number of chirps per frame is the same, increasing the inter-chirp time improves the velocity resolution but decreases the maximum unambiguous velocity. Therefore, the processing chain becomes more sensitive to motion. It is important to note that increasing the inter-chirp time through these parameters increases the total chirping window and reduces the total available processing time budget within the frame period. Hence, the user must ensure that the total chirping time and the required processing time can fit into a single frame period. These parameters also affect the average power consumption.
chirpComnCfg	chirpRampEndTime	

4.4 How to Configure the Angular Accuracy

Based on the sensor mounting option and the required target scene boundary, the user can adjust these angular limits accordingly. However, it is important to emphasize that the user should consider the antenna radiation patterns' physical limitations when configuring these parameters.

The angle resolution described in Section 4.1 is imposed by the total number of antenna elements used in the sensor array. As discussed in Section 3.1.3, the xWRLx432 is a single-chip radar device with 3 RX and 2 TX. Despite the physical limitations in the angle resolution imposed by the antenna pattern, the angle estimation accuracy can be improved by increasing the angular search grid in the FFT-based angle estimation step. This section provides a tuning approach for the angular accuracy requirements by configuring the related radar parameters.

Commands	Parameters	Tuning Guide
sigProcChainCfg	azimuthFftSize elevationFftSize	These parameters configure the angle search resolution in both azimuth and elevation domains. A larger value will improve the estimated azimuth/elevation accuracy but also increase the total processing time. The user should make a tradeoff and adjust the angle FFT sizes according to the memory, computation power, and average power consumption limits. Changing this parameter may result in the system running out of memory and MIPS. To fit the required

		processing time into a single frame period, the user must configure the frame period. When a small azimuth FFT size is configured, the azimuth interpolation block can be enabled (in interpolateAzimuth) can be enabled to improve the azimuth estimation accuracy.

4.5 How to Avoid Missed or False Detections

There are multiple parameters that affect when a missed motion/presence detection occurs while the boundary box is occupied by a single person or multiple people. If there are reflection points coming from the person, but the processing chain is unable to declare motion/presence, then the user should consider changing the following state transition parameters. However, it should be kept in mind that changing these parameters may also increase the chances of false detection.

Commands	Parameters	Tuning Guide
majorStateCfg **minorStateCfg**	pointThre1 pointThre2 pointHistThre1 pointHistThre2	If these point thresholds are lowered, fewer points will be needed to declare a motion/presence. Lowering these thresholds might be beneficial for longer ranges as fewer points might be detected from a person at larger distances from the sensor. On the other hand, lowering these thresholds may increase the false alarms.
	snrThre2 snrHistThre2	Further the distance of the target from the radar, the lower the SNR of the detected points. Lowering the SNR threshold will lower the cumulative SNR that is required to declare a motion/presence in the scene. On the other hand, lowering these thresholds may increase false alarms.
	histBufferSize	Increasing the value of this parameter may help in keeping the track alive for a longer time duration in case no points are detected in a few consecutive frames. When a larger buffer size is configured, the processing logic will look across a longer time window to see if enough points are detected with enough total SNR.
	major2minorThre minor2emptyThre	Increasing these thresholds will delay the state transitions between occupied to empty states to reduce the missed detections. In other words, increasing the value of this parameter may help in keeping the track alive for a longer time duration in case the detection is missed in a few consecutive frames. On the other hand, by lowering these thresholds, detections will be freed faster when the person exits from the scene.

5 References

[1] MMWAVE-L-SDK: xWRLx432 software development kit for low-power mmWave radar sensors, Texas Instruments. [Online]. Available: https://www.ti.com/tool/MMWAVE-L-SDK

[2] People counting demo: 'Tracking radar targets with multiple reflection points', Texas Instruments. [Online]. Available: https://dev.ti.com/tirex/explore

[3] MmWave training series, Texas Instruments. [Online]. Available: https://training.ti.com/mmwave-training-series

[4] E. Jacobsen and P. Kootsookos, "Fast, Accurate Frequency Estimators [DSP Tips & Tricks]," in IEEE Signal Processing Magazine, vol. 24, no. 3, pp. 123-125, May 2007

[5] M. E. Yanik and S. Rao, "Radar-Based Multiple Target Classification in Complex Environments Using 1D-CNN Models," in 2023 IEEE Radar Conference (RadarConf'23), San Antonio, USA, May 2023.

[6] mmWave DFP Interface Control Document (ICD), Texas Instruments. [Online]. Available: https://www.ti.com/tool/MMWAVE-DFP

[7] Sandeep Rao, "MIMO radar," Texas Instruments, Application Report, SWRA554A, July 2018.

[8] Vivek Dham, "Programming chirp parameters in TI devices," Texas Instruments, Application Report, SWRA553A, February 2020.

[9] Radar hardware accelerator user guide, SWRU527A, March 2018.

[10] IWRL6432BOOST BoosterPack™ plug-in module for single-chip low-power mmWave radar sensor. [Online]. Available: https://www.ti.com/tool/IWRL6432BOOST

[11] DCA1000EVM: Real-time data-capture adapter for radar sensing evaluation module, Texas Instruments. [Online]. Available: https://www.ti.com/tool/DCA1000EVM

[12] The mmWave sensing estimator, Texas Instruments. [Online]. Available: https://dev.ti.com/gallery/view/mmwave/mmWaveSensingEstimator