MIMORADAR OUTDOOR

Release 0.1

About this document

# Scope and purpose

MIMO radar outdoor application includes target detection, classification and positioning. Release 0.1 focuses on the fundamental framework of the whole process. The framework contains basic building blocks, when the radar is used to handle different tasks, these blocks can be reused.

In release 0.1, the target detection and classification is based on single channel, and the positioning is based on multiple channels.

This document serves as a guidance file to enable engineer to get started with the codes during development.

# Intended audience

[Internal] Radar engineers, machine learning engineers, dsp engineers in development phase.

# Contact person

Cheong Siew Lee ([SiewLee.Cheong@infineon.com](mailto:SiewLee.Cheong@infineon.com))

Table of contents

[About this document 1](#_Toc34836890)

[Table of contents 1](#_Toc34836891)

[1 Framework Overview 3](#_Toc34836892)

[1.1 FMCW radar principles 3](#_Toc34836893)

[1.1.1 Key configurations 4](#_Toc34836894)

[1.1.2 MIMO principle 6](#_Toc34836895)

[1.2 Blocks 8](#_Toc34836896)

[1.2.1 Preprocessing blocks 8](#_Toc34836897)

[1.2.2 Detectors and Classifiers 9](#_Toc34836898)

[1.2.3 Postprocessing blocks 9](#_Toc34836899)

[1.3 Tasks and blocks 10](#_Toc34836900)

[2 Radar Configurations 12](#_Toc34836901)

[2.1 Work flow 12](#_Toc34836902)

[2.2 Chirp configuration 12](#_Toc34836903)

[2.3 RF/Antenna configuration 12](#_Toc34836904)

[2.4 Base band configuration 13](#_Toc34836905)

[2.5 DSP configuration 13](#_Toc34836906)

[2.6 Postprocessing configuration 14](#_Toc34836907)

[2.7 Example configurations 14](#_Toc34836908)

[2.8 Other configurations 15](#_Toc34836909)

[3 Preprocessings 16](#_Toc34836910)

[3.1 2D FFT (RDM) 17](#_Toc34836911)

[3.2 DBF 18](#_Toc34836912)

[3.3 MTI filter 19](#_Toc34836913)

[3.4 CFAR filter 20](#_Toc34836914)

[3.5 Range and Doppler filters 21](#_Toc34836915)

[4 Detections and Classifications 23](#_Toc34836916)

[4.1 Detector and classifier selection 23](#_Toc34836917)

[4.2 Adding new detector/classifier 24](#_Toc34836918)

[5 Postprocessings 27](#_Toc34836919)

[5.1 Target Extraction 27](#_Toc34836920)

[5.1.1 DoA 28](#_Toc34836921)

[5.2 Target tracking 30](#_Toc34836922)

[6 Example Tasks 31](#_Toc34836923)

[6.1 Calibration 31](#_Toc34836924)

[6.1.1 Multi-path effects 34](#_Toc34836925)

[6.1.2 Real-time calibration in TDM MIMO 35](#_Toc34836926)

[6.2 Data collection 36](#_Toc34836927)

[6.3 Labelling 38](#_Toc34836928)

[6.4 Demo video with classifiers 43](#_Toc34836929)

[6.5 Stand-alone demo video without classifier 50](#_Toc34836930)

[References 51](#_Toc34836931)

[Revision history 52](#_Toc34836932)

# Framework Overview

To use MIMO radar for classification applications, a large amount of data need to be preprocessed before they are sent for trainings and for classifications. The preprocessing used for training and classification must be exactly the same. However during the solution development, there can be different methods to preprocess all the data, and for each method, there can be different implementations. Further, for different applications, e.g. low speed environments versus high speed environments, less target environments versus crowded targets environments, different processes may be preferred. Therefore, a framework is proposed here to create a basic flow and to contain processing blocks so that the development can be consistant and efficient especially when new related application request is raised.

The framework is written in Python, and it serves as a fast demo tool. The purpose is to select the correct processing blocks and do training, and then deploy them in embedded platforms.

## FMCW radar principles

This radar sensor is considered an active sensor since it receives signal sent out from designated transmitters. The transmitter generates modulated RF signals, e.g. frequency modulated continuous wave (FMCW) signals, pulsed radar signals, FSK Doppler signals, etc. Here FMCW modulation is used.

The details can refer to

* <https://www.infineon.com/cms/en/product/sensor/radar-image-sensors/radar-sensors/radar-sensors-for-consumer-and-iot/>
* AN553 *Position2Go-24GHz radar kit with BGT24MTR12 and XMC4700 32-bit ARM® CortexTM-M4 MCU for ranging and target position estimation* ([1])

The definitions of FMCW radar signals, chirp/frame and the working principles are clearly introduced in the above mentioned references.

In short, when the transmitted signal in one chirp (one ramp) hit a target, the microwave signal is bounced back and are received by the receivers. The time delay produced by the round trip of the EM signal is translated to a certain intermediate frequency (IF) spectrum by the FMCW mechanism in a linear manner. It means that once the spectrum generated by the target is detected, the corresponding time delay of the round trip can be obtained, further, the range of the same target can be calculated since the EM transmission velocity is also known (speed of the light). The process to get IF spectrum can be done using proper Fast Fourier Transformation (FFT) of the received ADC time domain data.

The above spectrum gives a range estimation of the target during one chirp, usually the signal is combined by Nc chirps, e.g. 32, 64, and these chirps are contained within one frame. Do FFT for each of the chirp will provide the range information for Nc times, by combining all chirps, the SNR may be increased, on the other hand, the results of the Nc chirps can also form a 2DFFT results to provide velocity of the targets. Therefore, after the 2DFFT of the ADC time domain signals, a range-doppler map (RDM) can be obtained for a single frame. An example is shown in Figure 1. In Figure 1, both camera image and radar rdm are shown. The horizontal line of the rdm represent the range and the vertical line represent the velocity. The bottom side of RDM has negative velocity means the target is leaving the radar, and the top side of the RDM has positive velocity means the target is approaching the radar. The center white line means stationary targets are observed by the radar at different ranges.

As shown in Figure 1, there are people walking away from the camera image near the gate, and it is also observed in the radar RDM with a velocity of around -0.8 m/s and a range of around 16 meters.

In this framework, the inference is based on RDM.

|  |
| --- |
|  |
| 1. Left side is the camera, right side is the RDM of the corresponding FMCW radar frame. |

### Key configurations

The RDM shown in Figure 1 consists of a number of range bins and Doppler bins as shown in Figure 2. The number of range bin equals to half of the sampling points in one chirp (num\_sample/2), and the number of Doppler bin equals to the number of chirps per frame (chirp\_per\_frame).

|  |
| --- |
|  |
| 1. Range bins and Doppler bins in one RDM. |

Here are some key configurations in terms of radar processing, not the RF configuration is not included:

* Chirp time and sampling rate

ADC with certain sampling rate Rsp samples the received signal consistently, so the chirp cycle time (including upchirp, downchirp, and the idle time) Tc and the Rsp determines how many sample points are available in one chirp. It in turn determines the effective size of FFT in range estimation and thus determines the number of range bins for one RDM.

The maxmum detectable speed is determined by the chirp cycle time:

where is the wavelength of the RF signal in the free space. It implies that for higher frequency radar, e.g. from 24 GHz radar to 60 GHz radar, to maintain the same highest speed detectable, the chirp time should be reduced roughly 2.5 times, at the same time, if the number of samples is fixed, say 64, then it means the ADC sampling rate should increases 2.5 times. Otherwise, the size of the formed RDM will be reduced, or the maximum measurable velocity gets smaller.

* Chirp per frame

Chirp signal is repeated multiple times within the frame, it provides the 2nd dimension of FFT, and generates the Doppler information. The number of chirp Nc in one frame equals to the number of Doppler bins. When the chirp time is determined, the maximum detectable velocity is fixed, and the velocity resolution is determined by the number of Doppler bins.

It can also be understood from RDM point of view, when the number of Doppler bins increases, each of the bin represent smaller velocity step:

For object classification applications, high velocity resolution is crucial. On the other hand, for embedded platforms, more chirps in one frame requests more memory and more processing power.

* Bandwidth

Besides the velocity resolution to construct a RDM, the range resolution should also be determined. The number of range bins has beem determined by the number of sample per chirp. And the range of each bin is determined by the total chirp bandwidth BW.

where is the speed of light. Then the maximum distance of this digital system will be times the number of range bins provided RF power and dynamic range are good enough to differentiate the signal out of the noises.

Now with the number of range bins (sample number) and Doppler bins (chirp per frame), RDM can be constructed in pixels. The step size of the range bin (range resolution) and Doppler bin (velocity resolution) then bring the digital pixels to physical world.

* SNR

Signal noise ratio defines how easily the target can be distinguished from the noise floor. The antenna FoV receives all in band signals including stationary targets, moving targets, clutters, signals from other electronic devices, reflections from the cover, etc. Also the nature of the signal (filter bandwidth) and the receiving chains all add up to the noise level.

Increasing the antenna gain and increase IF amplifer gain may not necessarily improve SNR, however, it definitely helps to see further (with a similar SNR).

To improve SNR, assume a fixed LNA, there are few options: increase data collection within one frame if the target speed is low, it also requires larger memory; reduce antenna to chip transition losses; use MIMO.

The configuration of radar used in Figure 1 is as following:

|  |  |
| --- | --- |
| **Chirp Bandwidth (MHz)** | **120** |
| **Sampling Rate (Msps)** | **1.5625** |
| **Upchirp Time (μs)** | **45** |
| **Chirp Time (μs)** | **180** |
| **Sample per Chirp** | **64** |
| **Chirp per Frame** | **256** |

According to the configuration, the constructed RDM has the following parameters: **range resolution** of 1.25 meters, note however, the real sampled bandwidth may be less than 120 MHz, and thus the achieved range resolution may be larger, that explains why the gate is actually 24 meters away from the radar but it shows a distance of around 20 meters in the RDM. Upchirp time is 45 μs, under a sampling rate of 1.5625 Msps, the number of samples obtained in one upchirp is 70 samples, and the processor only keep the 2nd sample to the 65th sample. Assume the range resolution is 1.25 meters, the RDM will have 64/2 = 32 range bins in the horizontal axis, this shows a **maximum measurable range** of 32 x 1.25 = 40 meters.

For velocity, the **maximum achievable velocity** is /4/180e-6 = 17.36 m/s (62.5 km/h). Due to the limited resources in XCORE, when the number of samples is fixed to 64, and for 4 channels (I or Q), the maximum number of chirp per frame is 256, note if both I and Q are recorded, this number is reduced to 128. Therefore, the +/- 17.36 m/s can be divided into 256 bins, and each bin has a **velocity resolution** of 17.36 x 2 / 256 = 0.136 m/s (0.489 km/h)

### MIMO principle

The above session about radar configurations are general and can be applied to different boards. The major difference between P2Go and MIMO radar board is the increased number of Transmitters (Tx) and receivers (Rx) to construct more channels to detect the same target, it provides a different measure of the target: spatial measurement. When the range is available, the position of the target can be determined.

When more antenna channels are available, each of the channels provides the RDM similar in magnitude but a time delay due to different propagation routes. When proper delay offsets are applied to different channels, after summation, the coherent signal can be enhanced while the non-coherent noise floor remains the same. This is considered as a gain in SNR. The more the channels are, the higher the gain is, and the higher angle resolution is. As shown in Figure 3, the target is received by a single narrow beam when the array size is large enough.

|  |
| --- |
|  |
| 1. A larger physical array produces narrower beam to enable higher angle resolution. |

Conventional multi-antenna channels are formed physically, and a large number of receivers are needed. To reduce the number of physical channels, a concept called virtual array is used as shown in Figure 4. Instead of using physical 7 Rx channels and 1 Tx channel, Figure 4 uses 4 Rx channels and 2 Tx channels. The two Txs send signals alternatively in time, and two frames of 4 channels are combined into one frame of 8 channels. The constructed RDM size is changed, the number of Doppler bins is halved in this case.

|  |
| --- |
|  |
| 1. Vitual MIMO array. |

The example in Figure 4 is further illastrated in Figure 5 where 1T2R (1 Tx and 2 Rx) and 2T4R (2 Tx and 4 Rx) are used to detect the same targets. The angle resolution with 2T4R is better compared to 1T2R. The improvement of SNR is shown in Figure 6.

|  |
| --- |
|  |
| 1. Targets detection using 1T2R and 2T4R. |
|  |
| 1. More channels (inclusive of virtual channels) improves the SNR. |

## Blocks

The purpose of the framework is to build up the foundamental blocks for radar AI related applications, including the data preparation for classifier (preprocessing blocks), AI processes including labeling tool, detector, classifier, and target level postprocessing blocks after classification. Each of the three major parts includes different approaches, when the API is carefully defined, these approaches are exchangeable and portable, and engineers from different areas can contribute to different parts by adding or changing these fundamental blocks. Configuration includes radar chirp configuration, processor (three processers) configuration, and RF configuration including antennas. It is important to ensure the configuration file remains unchanged for one application.

Figure 7 gives a general diagram of the whole process, note all codes are written in Python, but the blocks can always be changed to other languages for higher programming efficiency when following the same API. The current frameworks serves for demo purpose, is to investigate and train the possible best AI models for given radar configurations. The result of preprocess must be compared carefully between this framework and the deployed solution in order to apply the trained model. One example to move preprocessing into edge device is shown in Figure 8. Simultaneous preprocessing is available for edge device and it helps to further improve the efficiency. A careful comparison should done between the outputs from edge device and from the framework to ensure that the detector and classifier see the same inputs. The details of the processors will be explained in the following subsections and chapters.

|  |
| --- |
|  |
| 1. General diagram of the framework. |
|  |
| 1. Moving pre-processor from the demo (python) into edge device. |

### Preprocessing blocks

The raw data is not directly input to the trainer/classifier, instead, the raw data will be translated into RDM images as shown in Figure 1 and Figure 2. The RDM is then treated as a normal picture by the classifier, and thus AI techniques in computer vision can be used. Different filters can be applied to RDM before sending to the next stage. Preprocessing blocks mainly contain these filters. Figure 9 shows the details of the processors. Besides RDM, it also includes digital beamforming (DBF) to increase target SNR, moving target indication (MTI) filter to remove stationary targets, constant false alarm rate (CFAR) filter to adaptively remove noise and short-time-Fourier-transform (STFT) to handle RDM in adjacent time frames. The details are described in Chapter 3.

|  |
| --- |
|  |
| 1. Details of each part of the processors. |

### Detectors and Classifiers

Filtered RDMs are fed into detectors and classifiers for target detection and classification. Different AI frameworks and methods can be applied to the same data. Tradeoffs such as accuracy, speed, model size can be considered. As shown in Figure 9, current framework has two approaches: RCNN where the detection and classification are combined and CFAR for detection and XGBoost (or CNN) for classification. The choice of the methods depends on the edge device resources. More AI processors can be implemented.

The output of the AI processor are the bouding boxes of the identified targets in RDM, and the classification of the targets. With the bounding boxes and the known radar configuration, the distance and the radial speed of the targets can be obtained in the target extractor from post-processor.

The details are described on Chapter 4.

### Postprocessing blocks

Once the bounding boxes in the RDM frames are available from the AI processors, the post-processors extract the targets from each bounding boxes, find the direction of the targets, and tracks the targets. The extracted information of the targets include target range, speed, size, and class. Figure 10 shows an example of the extracted targets.

|  |
| --- |
|  |
| 1. Visuallization of the target level information on the right side. |

The details are described in Chapter 5.

## Tasks and blocks

According to Section 1.2, useful blocks are built and selected for different applications. The concept is shown in from Figure 11. Figure 11 shows all available blocks and the five main processes in radar applications: calibration, data collection, labelling, inference, and stand-alone application where no classification is needed. The blocks in the processors are available for different processes. The details are found in Chapter 6, and two examples of interence and stand-alone are shown in Figure 12.

In Figure 12, in inference mode, detector and classifier are enabled, and in stand-alone mode, cfar detector is used, and the result is directly passed to post-processor for target level processing.

|  |
| --- |
|  |
| 1. Overview of framework blocks and different processes. |
|  |
| 1. Inference and stand-alone radar applications with different blocks. |

# Radar Configurations

As shown in Figure 7, the same radar configuration is shared in one set of process including data collection, labelling and inference to ensure the data is processed in the same flow and the trained model is useful. Other parameters are also defined in the configuration files.

## Work flow

All configuration files are under .\configuration\

The work flow is shown in Figure 13. The detailed configurations for different applications or locations are stored in cfg\_optxx.py, and cfg\_main loads the correct option.

| Configuration.cfg\_main |
| --- |
| 1. From configuration.cfg\_optxx import mimo\_cfg |

For each of the five processes, always load the cfg\_main. This will make sure all processes share the same definition of RDM parameters including RDM size and bin resolutions.

| Calibration/Data collection/labelling/inference/stand-alone |
| --- |
| 1. From configuration.cfg\_main import RadarConfigure 2. cfg = RadarConfigure() |

|  |
| --- |
|  |
| 1. Configuration files arrangement. |

## Chirp configuration

Chirp configurations are stored in cfg\_optxx.py. Definitions such as chirp bandwidth, center frequency, number of samples in one upchirp, number of chirps in one frame, chirp time are given. Some of the parameters are used to calculate the radar specifications such as range and Doppler resolutions in the same file.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.bw = 120e6 2. … |

## RF/Antenna configuration

Antenna configurations are stored in cfg\_optxx.py. Physical relative locations of the antennas are recorded for this 2T4R board. It is crutial to use the correct locations to create virtual arrays for MIMO applications such as positioning and beamforming.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.tx\_antenna1 = 0.0186 2. self.tx\_antenna2 = 0 3. … |

The physical locations will be translated into electrical length when the wavelength is available. Further, the code enables to choose which Tx antennas and Rx antennas are used, and what is the order to use Tx antennas in TDM MIMO applications.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.tx\_enable = [0] 2. self.rx\_enable = [0, 1, 2, 3] 3. self.tx\_in\_use = “1” 4. self.rx\_in\_use = “1234” |

Therefore, once the PCB is fabricated, the physical dimensions can be input. And in applications, the enabled Tx and Rx can be chosen according to request.

The calibration matrix is also stored in the configuration file.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.ang\_cal\_mtx = […] 2. self.mag\_cal\_mtx = […] |

## Base band configuration

Baseband configurations mainly includes ADC information such as ADC bit number and whether it is signed or unsigned. Also the IQ information is included here.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.iq\_mode = 0 2. self.iq\_setup = “i” |

When both I/Q channels are used, SNR can be improved compared to a single I/Q channel. I/Q channels are combined in time domain before DSP.

## DSP configuration

DSP configurations mainly cover preprocessing parameters. The link between Radar configuration and DSP is the RDM size. Zero padding is not considered in Release 0.1. For preprocessing, DSP configurations defines whether to use MTI filter or CFAR filter:

| Configuration.cfg\_opt1 |
| --- |
| 1. self.mti\_flag = False 2. self.cfar\_flag = False |

When CFAR is used, the CFAR parameters are also defined in this part:

| Configuration.cfg\_opt1 |
| --- |
| 1. self.cfar\_guard = 5 2. self.cfar\_length = 8 3. self.cfar\_scale = 5 |

These parameters will be used when CFAR algorithm is used for each RDM.

Note these two filters are applied to the raw RDM. During detection, the CFAR function is directly called.

## Postprocessing configuration

Postprocessing configurations in the configure file mainly reflects the area of interest and the installation of radar. After the target is detected, the program will only search within the defined area in a defined step. Because the radar can be installed in different heights and orientations, these position information need to be compensated in the final output as well.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.distance\_min\_m = 1.5 2. self.distance\_max\_m = 45 3. self.azimuth\_fov\_deg = [-60, 60] 4. self.azimuth\_res\_deg = 2.0 5. self.azimuth\_offset\_deg = 20 6. self.elevation\_offset\_deg = 0 |

Note these information may also be useful for RDM interpolation especially when elevation offset is large.

## Example configurations

By default, two options are provided for different applications: short range (40 meters) for target classification purpose and middle range (60 meters) for target detection purpose. Foe classification, higher velocity resolution is necessary while for detection only, higher maximum detectable velocity is preferred. The main limitation comes from the RDM size, which is limited by the processor memory. Both options fully used the available memory size (to have 256 chirps per frame and 64 samples per chirp). When the memory for this process is fixed, for the same number of chirps in one frame, higher velocity resolution means a lower maximum detectable speed. The major difference of the two options thus is coming from the RDM resolution.

| Configuration.cfg\_opt1 |
| --- |
| 1. self.bw = 120e6 2. self.sample\_per\_upchirp = 64 3. self.chirp\_per\_frame = 256 4. self.up\_ramp\_time = 45e-6 5. self.down\_ramp\_time = 45e-6 6. self.chirp\_interval = 90e-6 |
| Configuration.cfg\_opt2 |
| 1. self.bw = 80e6 2. self.sample\_per\_upchirp = 64 3. self.chirp\_per\_frame = 256 4. self.up\_ramp\_time = 45e-6 5. self.down\_ramp\_time = 25e-6 6. self.chirp\_interval = 90e-6 |

## Other configurations

There are other configurations which are not linked to hardware and DSP, but the MCU and display. These are directly defined in .\configuration\cfg\_main.py. This can also isolate unnecessary information needed from customer in cfg\_opt files.

| Configuration.cfg\_main |
| --- |
| 1. def config\_recording(self): 2. … 3. def config\_usb(self): 4. … |

USB configuration defines the size of data in bytes to be recorded in one frame.

# Preprocessings

The raw data is not directly fed into the detectors and classifiers. The AI frameworks are modified from computer vision applications, thus the raw data needs to be visualized first and then fed into the AI framework. As introduced in Chapter 1, the raw data will be visualized as RDM, and the purpose of preprocessing to apply appropriate filters, so that the image seen during labeling and inference has less clutter and higher SNR, and the targets can be better recognized. A basic flow is shown in Figure 14. Note there are several different blocks in the preprocessor, for both training and inference, every parameter of the preprocessor must be the same.

In current framework (Figure 14), the preprocessing blocks are used to:

* generate the images, and
* provide filtering for a better segmentation.

The purpose to differentiate them is to provide as much information as possible to the labelled samples. In this stage, efforts to reduce inter-bin leakage (window type), SNR improvement (DBF), and clutter filtering (MTI) can be used to build the “raw” images. After that, labels and inference provide the detection, which usually consists smaller cluster of pixels with the help of other filters like range Doppler filter and CFAR filters. Segmentations and binarizations are done after the filtering. Once these pixels are found, the program goes back to the “raw” images, and collect the raw data and bounding box from it instead of from the filtered data. In such a way, the classification training is based on the “raw” images.

|  |
| --- |
|  |
| 1. Basic flow to enable radar AI based detection and classification. |
|  |
| 1. Preprocessing blocks arrangement |

Figure 15 shows how the preprocessing blocks are usually used in the codes. Different blocks (block\_rdm, block\_td\_mti, block\_cfar) are added into preprocessing.blocks, and preprocessing.blocks is loaded into main configurations as an attribute. For applications such as labelling, inference, and stand-alone, the whole configurations are loaded together.

## 2D FFT (RDM)

An example of the radar raw data with 10 chirps is shown in Figure 16. The details of the FMCW signal can be found in Chapter 1 and [1]. FFT is taken for each of the chirp, and thus a window is applied to these 64 samples. The window in this case is chosen to be hannning window to reduce signal leakage for further spectrum bins. Different windows can be chosen for different purposes. But as stated in Figure 14, for one complete process, the windows for all labelling and inference should be aligned.

|  |
| --- |
|  |
| 1. Time domain signals and preprocessing before RDM: (left) raw ADC sampled signals, (center) hanning window, (right) windowed ADC data to feed into FFT. |

The code details can be found in

| Preprocessing.block\_rdm |
| --- |
| 1. def cal\_range\_fft(): 2. … |

Similarly, the window is also applied to Doppler as stated in

|  |
| --- |
| Preprocessing.blocks |
| 1. class dspProcessor: 2. def \_\_init\_\_(): 3. … 4. self.range\_win = np.array([np.hanning(self.spc)]) 5. self.doppler\_win = np.array([np.hanning(self.cpf)]) |
| Preprocessing.block\_rdm |
| 1. def cal\_rdm(): 2. … |

The size of the the window is changed from the number of samples per chirp to the number of chirp per frame. Time domain signal to 1D FFT is complex number, and 1D FFT results give the range information. The range bin size is halved compared to the samples due to the repetition in spectrum. Note if I and Q channels are used, additional SNR can be obtained in spectrum. 1DFFT results of 256 chirps are passed to 2DFFT for RDM computation.

A sample RDM is shown in Figure 17 by using visualization tool:

| Postprocessing.viz |
| --- |
| 1. def cv2\_show\_rdm(): 2. … |

|  |
| --- |
|  |
| 1. Sample single channel RDM result. An object is moving away from the radar in a low speed (~-2m/s) |

The image in Figure 17 is considered as a “raw image”, and the information loss is considered “small”.

## DBF

To further improve the SNR, digital beamforming can be applied in the receivers as explained in 1.1.2. Single antenna can see a broad field, usually the FoV is the 3dB beamwidth of a single antenna. In this radar, the FoV is for single antenna is around 80 degrees [1]. Beams are generated by summing all channels in time domain:

| demo.make\_demo\_pics |
| --- |
| 1. for ich in range(len(vec)): 2. adc\_multi[ich] = adc\_multi[ich] \* complex(…) 3. np.sum(adc\_multi, axis=0)/len(vec) |

The implementation shows that the beams are formed in ADC time domain level. After the signals are summed as in code line 052, it is sent for 2DFFT and generate a RDM just like a single channel. The difference is, because the noise from different channels are incoherent, so the distribution remains the same, but the intensity is suppressed as in 052. On the other hand, the useful signal intensity remains the same because the signals in different channels are coherent. As shown in Figure 3, when beams are formed, the beamwidth gets narrower and the beams are pointed to different directions. When the target is in the direction, then the SNR of this specific target can be improved. This direction determines vec as in code line 059.

| demo.make\_demo\_pics |
| --- |
| 1. def get\_beam\_vec\_1d(): 2. beam\_angles = np.array([0, -17, 17]) \* np.pi / 180 3. … |

The beam direction is defined in the numpy array. In this case, three beams will be generated and they are looking at 0°, -17°, and 17°, respectively. Figure 18 shows a beam looking at 0°of Figure 17. It is obvious that the SNR is improved for the same scenario.

Figure 19 shows the three radar images to capture pedestrians walking on the left region. It is clear that the left side beam has a larger signal intensity compare to the other two beams.

|  |
| --- |
|  |
| 1. Sample RDM result in a MIMO beam. The noise has been suppressed compared to Figure 17. |
|  |
| 1. Three radar beams generated to observe the left side pedestrians. Higher SNR will help for target detection. |

## MTI filter

Stationary objects are clearly shown in the RDM images, from Figure 14, these images will be sent for inference. In circumstances/algorithms where these targets at the center of the images are detected unintendly, extra efforts may be needed from detection algorithms. Another approach is to apply moving target indication (MTI) technique to the images.

|  |
| --- |
|  |
| 1. Illustration of the MTI filter in preprocessing.block\_td\_mti.py. |

The approach can be implemented in preprocessing blocks. Here in release 0.1, time domain MTI filter is implemented. It is basically an implementation of a low-order low-pass FIR filter to filter out the DC signals. When higher order is applied, a steeper transition or a bandpass filter can be achieved [2]. The FIR high pass filter is implemented such that the chirps are shifted one chirp time, and minus the original chirp. The last sample is shifted to the first chirp. The illustration is shown in Figure 20. As a result, in Doppler spectrum, the DC (speed = 0) signal is filtered out. The implementation is in below. Note other more complicated and comprehensive filter collections can be added in the same format.

| preprocessing.block\_td\_mti |
| --- |
| 1. def td\_mti(): 2. … |

One example is shown in Figure 21. By adding MTI filters, the intensity near zero Doppler spectrum are suppressed and the targets get less significant, meanwhile, the stationary clutters are fully removed. For **INDOOR** applications where the maximum speed is not the highest priority, this filter can be very useful for low speed movement with a specific chirp configuration.

|  |
| --- |
|  |
| 1. Moving targets without (left) and with (right) MTI filter. |

## CFAR filter

Radar systems produce random noises, in some frames, these noises may be detected as targets. The operation temperature of the radar gets higher, the noise floor raises. Enviroment also produces noises, for example, when there is a nearby strong reflection, the radar system may produce noisy images.

Besides the noises, the useful signal, determined by the radar cross section (RCS) of the targets, may vary with targets’ facing directions. The variation can reach more than 20 dB for a small change of looking angle. Furtherm It is natural for a fading in signal intensity when the targets are moving further.

The task is to always differentiate the correct targets from the jumping noises by using threshold. When a fixed threshold is used, it may fail to detect further targets or targets with small RCS (e.g. pedestrian) when the threshold is too high, or it may detect noises frequently and produce flase alarm when the threshold is too low. Therefore, an adaptive threshold (filter) to produce constant false alarm rate is necessary. Figure 22 illustrates the one dimentional CFAR implementation, note a two-dimentional CFAR where both range and Doppler are considered can also be implemented with a higher computational burden. In the implementation, the purpose is to determine whether there is a target in the bin under test (BUT) with filled green color and red outline, and the noise is adaptively determined by the surrounding green training bins with black outlines. The noise is the summation of the training bins multiplies by a scaling factor. If the value in BUT is higher than the scaled noise, then it means there is a target at this range and speed, otherwise, the BUT is empty.

|  |
| --- |
|  |
| 1. Illustration of 1D CFAR in range bins. |

The implementation is as below. There are other cfar algorithms can be added into this block. ca\_cfar2 flattens the 2D array into 1D array to speed up the computation.

| preprocessing.block\_cfar |
| --- |
| 1. def ca\_cfar(): 2. … 3. def ca\_cfar2(): 4. … |

And the main parameters are set in the configuration files as stated in 2.6, code lines 019-021. The number of training bins is the cfar\_length, and the number of guard bins is the cfar\_guard, the noise scaling factor is the cfar\_scale.

One example is shown in Figure 23. By applying the CFAR filter, the targets are more significant and the noises at the edge of the original image is removed to avoid false alarm.

|  |
| --- |
|  |
| 1. Moving targets without (left) and with (right) CFAR filter. |

## Range and Doppler filters

One additional filter to add is a range/Doppler window (filter) to exclude the targets detected outside of the filter. This filter is added based on the installation location and relative facing between the targets and the radar. In practical labelling and inference, it is found that in some closely spaced region, the signatures of different types of targets are very similar and it may change the model significantly. Therefore, this filter can add flexibility to those applications where a small region can be excluded.

| demo.make\_demo\_pics |
| --- |
| 1. def load\_detector(): 2. … 3. rd\_filter = [5, 32, 0.6] |
| radar\_label\_sample.label\_tool\_v0.1 |
| 1. Cfg = RadarConfigure() 2. rd\_filter = [5, 32, 0.4] |

|  |
| --- |
|  |
| 1. Example of RDM when a car (left) and a truck (right) is near the radar. The shapes are quite similar. |

# Detections and Classifications

The focus of this chapter is how to integrate AI processor into the framework. The details of the AI models, like how to train the model, and the model layers and parameter setups, are out of scope of this document.

Once the data have been properly preprocessed, an image (without CFAR filter or range Doppler filter) is obtained. This image can be used for labelling or inference (both tasks are introduced in Chapter 6). For inference, when an image is provided, the next steps include detection and classification. As shown in Figure 9, there are many potential methods and algorithms, and this framework is to provide a seamless interface to use all implemented AI processors for performance evaluation in demo stage. Once a good method is found, further deployment can be considered. Release 0.1 has two sets of AI processor: RCNN where detection and classification are combined and CFAR + XGBoost where detection and classification are separated. The outputs are the target information in this image, and the formats are the same: bounding boxes location, size, and predicted class. Figure 25 shows the block arrangement in the framework. Demo.make\_demo\_pics.py uses recorded offline data, and the choice of detector and classifier is defined in this file.

|  |
| --- |
|  |
| 1. Detection and classification blocks arrangement. |

## Detector and classifier selection

The detector and classifier selection is done in DemoOffline class initialization

| demo.make\_demo\_pics |
| --- |
| 1. class DemoOffline: 2. def \_\_init\_\_(): 3. … 4. self.work\_opt() |

There are three work options, in work option 0, raw images are shown without any label. In work option 1, the code is looking for manual labels, and show the bounding boxes and labelled classes in the images. And in work option 2, the detector and classifiers should be used. And the work option is selected in the main code:

| demo.make\_demo\_pics |
| --- |
| 1. if \_\_name\_\_ == “\_\_main\_\_”: 2. WORK\_OPT = 2 |

In this case, the detector and classifier are assigned in the following function:

| demo.make\_demo\_pics |
| --- |
| 1. class DemoOffline: 2. … 3. def work\_opt(): 4. … 5. elif work\_opt == 2: 6. … 7. self.load\_detector(det\_type=”rcnn/cfar”) 8. self.load\_classifier(cls\_type=”rcnn/xgboost/non”) 9. def load\_detector(): 10. … 11. def load\_classifier(): 12. … |

## Adding new detector/classifier

New detectors and classifiers can be added under the .\detectors\ and .\classifier\ folders. All newly added blocks should have the same interface as shown in Figure 26. Detector handle the complete image, find the detected bounding boxes, and then classifiers predict classes within the provided bouding boxes. If the detector and classifer is combined, like RCNN, then the input is the image and the output must be the bouding boxes and enclosed classes. Output of bounding box is a must for radar target extraction.

|  |
| --- |
|  |
| 1. Input and output of detectors and classifiers. |

If the AI processor is developed separately, like rcnn in .\gluon\_faster\_rnn\, a wrapper is needed and it is located in .\classifier\ to align with Figure 26. The initializations can be quite different for different AI processors. For example in rcnn, setups for network, context (GPU/CPU), bouding box proposal and corresponding parameters are all loaded in initialization stage.

| classifiers.get\_rcnn\_label\_det\_cls |
| --- |
| 1. class RcnnClassifierEnd2End: 2. def \_\_init\_\_(): 3. … |

Similarly, for CFAR detector and XGBoost classifier, setups of filters and models are loaded in initialization stage.

| detectors.get\_bb\_cfar |
| --- |
| 1. class dspProcessor: 2. def \_\_init\_\_(): 3. … |
| classifiers.get\_xgboost\_label\_cls |
| 1. class XgboostClassifier: 2. def \_\_init\_\_(): 3. … |

The input of the all AI processors should be aligned, it is an rdm returned from preprocessing:

| demo.make\_demo\_pics |
| --- |
| 1. dsp = preprocessing.blocks.dspProcessor 2. rcnn\_channel = dsp.process\_rcnn\_channel(adc\_data) |
| preprocessing.blocks |
| 1. class dspProcessor: 2. def process\_rcnn\_channel(): 3. … 4. return rdm\_filter\_rcnn |

The obtained rdm is raw rdm, which means only certain windows, DBF and MTI can be applied here, and thus for inference, stand-alone and labelling, the configuration for the CFAR flags should be False. The digitalization of rdm is done within different AI processors if necessary.

| demo.make\_demo\_pics |
| --- |
| 1. bbs = self.detector.get\_bb(rcnn\_channel) |
| detectors.get\_bb\_cfar |
| 1. class dspProcessor: 2. def get\_bb(): 3. … 4. rdm = viz.rdm\_cvt\_uint8(rdm) |

viz is a visualization tool under postprocessing. From the above, it is clear when a new AI processor is provided, its input is one raw rdm frame. Filtering should be within the AI processor itself as shown in Figure 14. The interface between detector and classifier is the bounding boxes, which followed the definitions in the labelling tool. Each of the frame may have N proposed bounding boxes and they are conveyed to the classifiers as a list:

1. [[[bb1]], [[bb2]], …, [[bbN]]]
2. bb1 = [x\_c, y\_c, width, height]

The reason to have bb1 (code line 105) within another layer of list is because there may have multiple rules for bounding box generation. Each of the bounding boxes contains four values: the x, y of the center point and the width and height of the bounding box. These values are all **normalized** to the rdm size.

Classifier produces classification results with the bounded data by the same wrapped function: get\_labels.

| demo.make\_demo\_pics |
| --- |
| 1. if self.detector\_type == “rcnn”: 2. labels = self.ml\_framework.get\_labels(rcnn\_channel) 3. if self.detector\_type == “cfar”: 4. if self.classifier\_type == “xgboost”: 5. labels = self.ml\_framework.get\_labels(rcnn\_channel, bbs) |

And the output labels of each frame

1. [[label1], [label2], …, [labelN]]
2. label1 = [ID, class, x\_c, y\_c, width, height]

If no classifier was used and the targets are detected, the same format is used and class = -1.

| demo.make\_demo\_pics |
| --- |
| 1. if self.detector\_type == “cfar”: 2. if self.classifier\_type == “non”: 3. labels = [] 4. for bb in bbs: 5. labels.append([0, -1, bb[0][0], …]) |

Make sure all newly added AI processors have the same input and output.

# Postprocessings

For one frame of ADC data, one image is built based one single channel (ideally this channel should have the highest SNR among other channels), and the image is used for target detection and classification, bounding boxes are generated.

The postprocessing further extract target level information from the bounding boxes in a more stable way by using tracker. The flow is shown in Figure 28.

|  |
| --- |
|  |
| 1. Postprocessing blocks arrangement. |
|  |
| 1. Postprocessing flow from bounding boxes to target level information. |

## Target Extraction

As shown in Figure 28, after AI processor, bounding boxes (labels) in each frame are conveyed to the postprocessor. All images (rdms\_all) are also sent for target extraction as shown in Figure 28.

| demo.make\_demo\_pics |
| --- |
| 1. if self.target\_pos: 2. tgts = extract\_tgt(rdms\_all, labels) |

| postprocessing.extract\_tgt |
| --- |
| 1. Class Tgt: 2. … 3. self.det\_in\_label = self.find\_det\_in\_tgt() 4. … 5. def find\_det\_in\_tgt(): 6. … 7. thresh = 0.3 8. … 9. def extract\_class(): 10. … 11. def extract\_class\_score(): 12. … 13. def extract\_average\_speed(): 14. … 15. def extract\_center\_x(): 16. … 17. def extract\_center\_y(): 18. … 19. def find\_point\_doa(): 20. … 21. doa = doa\_bf(point\_all\_channel, va, doa\_list) 22. … 23. def extract\_width(): 24. … 25. def remove\_ghost(): 26. … 27. def compare\_width(): 28. … 29. def compare\_length(): 30. … |

A bounding box gives a rectanglular area of bins. However, not all bins in the rectangle contain reflected signal. These bins with less signal intensity helps in target classification, but thet are not needed for target extraction. A threshold is applied to determine the useful bins within this box (code line 127). When these bins are identifiled, the direction of the reflection in each bins are found (code line 139). We assume one bounding box only includes one specific target, thus all bins have similar directions. For bins located very far away, they are identified as ghosts and these bins are removed from the list of useful bins (code line 145).

The useful bins now have range information and angle information, and can be represented in x-y axis (at the right Figure 10). Based on these information, the following parameters of the target can be obtained: width, length, speed and location (x and y).

The class and class score information is directly passed to the target level information. A filter can be added to give up low score targets here.

### DoA

From the above section, a critical step is to find the angle information of useful bins in each bounding box. The angle information further helps to estimate the x-y position, x-y speed, and target width. This is realized by direction of arrival (DoA) block. In release 0.1, beamforming method is used for DoA. The basic concept is illustrate in Figure 29. When the target is in the far field, the reflection can be considered as a plane wave, and the wave front will be received by the receiver array. After the calibration (as introduced in Chapter 6), all signals have the same amplitude. Because the wavefront reaches different antennas in different times (as in Figure 29, signals reaches antenna 2 earlier compared to antenna 1), there is a signal delay between adjacent antennas. The delay can be compensated with a phase change, when a proper phase change has been applied to the signal in antenna 2 as shown in the right figure, the signals in the two channels can be coherently summed and the signal is amplified by a factor of 2. This phase shift applied is determined by the direction of the target. If the applied phase shift is not correct, as shown in the center figure, the combined signal strength can be even reduced.

|  |
| --- |
|  |
| 1. Basic concept of DoA method. The plane wave is received by all rx antennas, only when the signals are coherently added, the received signal can be amplified. |

Once the best phase shift ϕ to produce the highest amplitude is found, the target direction θ can be calculated

The implementation is as follows:

| postprocessing.block\_doa |
| --- |
| 1. def doa\_bf: 2. … 3. weight\_win = np.array([1, 1, …]) 4. … 5. weights = weight\_win \* complex(angle) 6. intensity = np.sum(point\_all\_channel \* weights) 7. … |

A weight window can be applied to the signals at different channels, when the array size increases, the window allows an elimination of side lobes by sacrificing the angle resolution. Here all weights are set to one so that the highest angle resolution using beamforming method is used. The angle of interested is defined in the configuration file as in code line 024 - 025, for each of the angle, a vector for all channels are calculated (code line 155), and the overall summation of the signal is calculated in code line 156. Then the angle with the maximum signal strength is found as , since the other parameters and are given, the DoA is calculated.

The output is a list of single target with a fixed length.

1. [[target1],
2. [target2],
3. …
4. [targetN]]

Each of the target contains nine elements:

1. [target1] = [target ID,
2. target class,
3. target class score,
4. track level,
5. target y-axis speed,
6. target width,
7. target length,
8. target center x,
9. target center y]

## Target tracking

Target tracking takes one more measure of detected objects. It contains a tracking level to ensure the objects have been correctly identified in continuous frames. The tracking is based on the location and the speed of the targets. Based on these two information, the position of all detected targets in the next frame can be predicted, if the measurement in the next frame matches the prediction, then this target is successfully tracked. The trakcing level will be increased by 1. On the other hand, if a noise was detected in the current frame with a random postion and speed, the possibility of a noise detected in the predicted location is low. And the noise will not be tracked.

|  |
| --- |
|  |
| 1. Target tracking. |

Thus, AI processor can have wrong detections as shown in Figure 30, however they will not tracked. When a filter is set to ignore lower tracking level, these wrong detections are eliminated. The implementation is as follows:

| postprocessing.block\_track |
| --- |
| 1. class tracker: 2. … |

# Example Tasks

This chapter focuses on the five major tasks that needs to do during the whole radar AI outdoor target classification application.

## Calibration

In principle, any radar set should have its own calibration matrix. Any change in the signal link from radar cover (including the change of distance from antenna board to cover), antenna, RF chain, IF chain and all the way to ADC will need a calibration before using. A basic calibration of boresight target is shown in Figure 31. More advanced calibration will include other positions of targets (from other directions). The concept is to capture signals in different channels using targets with known positions and transform these signals into calibration matrix. In this case, the target is placed in boresight of the radar antenna board, when there is a reflected signal from the target, the generated plane wave should arrive the antennas at the same time. However, since there is a non-ideal cover, there will be some delays of some channels. Futher, the magnitude may also be affected. The received signal of different channels then experienced RF chain, IF chain and ADC, each of the channel will produce different amplifications (attenuations) and time variants, and all these imbalances are accumulated at point 1’ and 2’ for channel 1 and 2. Since the target is stationary, 1D-FFT is applied to signals received at 1’ and 2’ to obtain spectrums. The magnitude and phase are found through the spectrums to consider the above mentioned accumulated imbalances. Note time variants for small bandwidth can be translated into phase changes. And these magnitude and phase are used to generate the calibration matrix.

|  |
| --- |
|  |
| 1. Calibration principle. |

In .\calibration\ folder, there are a few files start with calibration\_.

* Data record: .\calibration\calibration\_mimo2go\_record.py

| calibration.calibration\_mimo2go\_record |
| --- |
| 1. if \_\_name\_\_ == “\_\_main\_\_”: 2. cfg = RadarConfigure() 3. cfg.show\_plot = True 4. cfg.show\_channel = 3 5. cfg.recording = False 6. cfg.filetime = 10 |

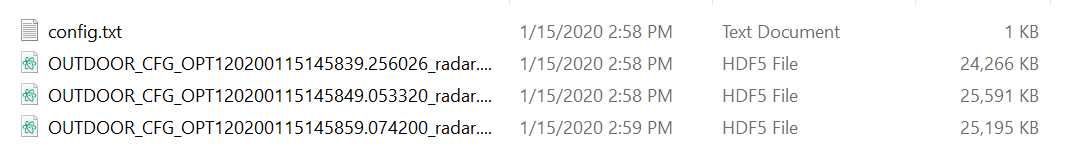
The major useful setups are the code lines from 174 to 178. Make sure the correct configuration file is chosen.

Before a recording, it is recommended to show plot without recording. It gives a real-time RDM plot of the chosen channel. The purpose is to verify if the system is working and also verify whether the parameters are correct: since the position of the target is known, the distance can be translated to the correct bin number when a correct bandwidth is used. Different channels can be tested by changing the cfg.show\_channel parameter, and the channel starts from 0.

Once the system has been tested, enable the recording. Data will be recorded under

.\calibration\data\_collected\

The amount of data of each file is also determined by the cfg.filetime (in seconds).



* Generate calibration matrix: .\calibration\calibration\_main.py

In each of the data recording folder, a config.txt file (json format) is used to record basic radar information needed for the calibration.

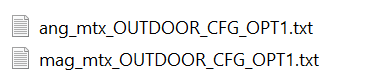
| calibration.calibration\_main |
| --- |
| 1. if \_\_name\_\_ == “\_\_main\_\_”: 2. … 3. cfg = json.load(f) 4. cfg = add\_additional\_cfg(cfg) |

Additional information is still needed for calibration such as target position (to be input for different calibrations), which was not included in the radar configuration itself.

| calibration.calibration\_main |
| --- |
| 1. def add\_additional\_cfg: 2. … 3. cfg[“cali\_target1”] = (10, 0, 0) |

As stated in code line 185, the target is placed 10 meters away with both elevation and azimuth angle of 0°. Since the bandwidth is already loaded in the configurations, the range resolution can be obtained, thus the correct range bin can be obtained.

The calibration will generate calibration matrix in .\calibration\Cal\_mtx\



Since there are 10 seconds of data recorded, and the calibration matrix is based on 1D-FFT of each single chirp, there can be more than 10 thousand FFT results for the same spectrum. The matrix are generated based on the average value of the last 100 spectrums.

| calibration.calibration\_main |
| --- |
| 1. class Calibrator: 2. def \_\_init\_\_(): 3. … 4. self.gen\_cal\_mtx(file\_prefix, num\_cal=100) |

The statistical information is also visualized for the calibration data under:

.\calibration\calibration\_plots

It shows how magnitude and phase vary at the allocated range bin over all chirps. The calibration has one channel as a reference on the left side. As shown in Figure 32 and Figure 33, if there is are big jumps or variations on the distribution, it means the selected range bin may be incorrect. These figures are the results from open space measurement as shown in Figure 34, the variation will be much smaller if the measurement is done in an anechoic chamber room (Figure 35).

|  |
| --- |
|  |
| 1. Statistical figures of magnitude (channel 0 as reference, channel 1 and its histogram). |
|  |
| 1. Statistical figures of phase (channel 0 as reference, channel 1 and its histogram). |
|  |
| 1. Open space measurement setup. |
|  |
| 1. Anechoic chamber room setup. |

### Multi-path effects

The major differences between the two measurement environments in Figure 34 and Figure 35 are:

* Chamber room does not have other targets in the FoV.
* Multi-path effects are eliminated within chamber room.

The illustration of multi-path is shown in Figure 36. In an open space, when the radar signal is reflected from the target, it is radiated to all directions. The signal that directly return to the radar is just one of the directions, and this signal provide the most accurate measure of the target. There are also lots of other signals are bouncing back via other paths like reflections from the ground or other targets, and these signals can also be captured by the radar but the radar does not differentiate them from the direct reflection. This is called multi-path effects. The travelling route is a bit longer, and this longer distance is translated to a slightly different spectrum. If the range resolution is not high enough, these spectrums are squeezed into the same range bin, and they all contribute to the magnitude and phase information. Again, however, the radar will not know.

|  |
| --- |
|  |
| 1. Multi-path effects combined with antenna mutual coupling. |

Suppose all antennas receive the same combinations of signals, means even there is multi-path effects, we assume they are the same to all antennas. When antenna receive signal, there is a concept called phase center of the antenna, it acts like a “source generation point” of the received signals. When the phase centers of all antennas are in the same horizontal line, it will not tell the difference between the multi-path signals from one vertical plane; however, if the phase centers are not in the same horizontal line, the multi-path signals from different elevation angle will already produce phase difference for each antenna. And unfortunately, such shift of phase center in elevation is common in mmwave antenna arrays due to stromg mutual coupling [4], and usually such information is unknown.

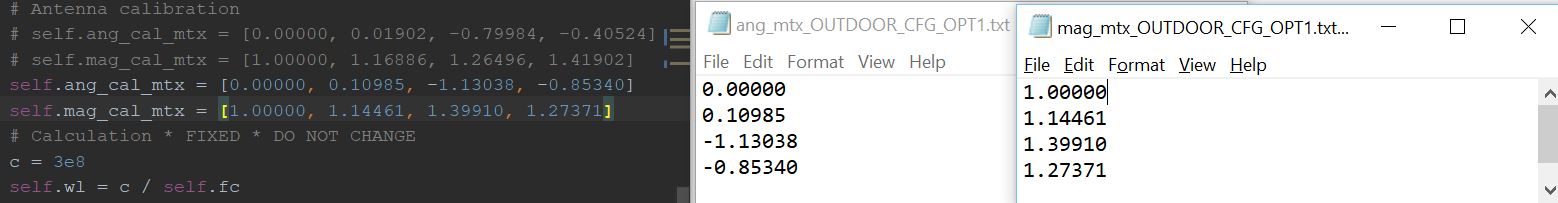
Multi-path effects will affect the calibration and a straightforward way is to remove the multi-path in the calibration stage in anechoic chamber where the bouncing signals are absorbed and the radar only receives direct reflected signals.

It also brings the topic that for AI applications, the multi-path effects may play a role in target signatures.

### Real-time calibration in TDM MIMO

This subsection is to show how the calibration results can be applied.

Once the angle and magnitude matrix are available, they are input into configuration files as shown below. In this case, 1T4R is used and the calibration produces 1 x 4 vector for each of the matrix.



| configurations.cfg\_opt1 |
| --- |
| 1. class mimo\_cfg: 2. def \_\_init\_\_(): 3. … 4. self.cal\_mtx = np.exp(1j \* ang) \* mag |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def cvt\_adc\_data\_to\_list(): 4. … 5. radar\_raw[ch] = raw[“t1\_r2\_i”] /   self.cfg.radar\_cfg.cal\_mtx[ch] |

When there is only one Tx, the above calibration will be enough. However if there are multiple transmitting antennas to form a virtual array as shown in Figure 4, one more step is needed.

As shown in Figure 37, there are four physical receiving antennas in grey and two transmitting antennas in black and red. The transmitting antennas transmits alternatively. At frame 1 at the left side, black transmitting antenna is transmitting, the system sees one target, at frame 2 at the right side, red transmitting antenna is transmitting and the system sees the same target with the same speed. The time difference between frame 1 and frame 2 is less controlled because two transmitters may not be synchronized in layout. Also the radiated power may be different. Thus, although the receiving antennas are receiving the same reflection from the same target, the obtained phase and magnitude may be different. The 4 channels in frame 1 and in frame 2 is facing a step discontinuity when they are combined into a larger virtual array.

One solution can be very careful synchronization. In our case, we design the array in a way that one element position is shared by the two arrays as shown in Figure 37. This is valid when the target in the two adjacent frames does not change too much, and this is usually the practical case in the applications. As a result, the last channel in frame 1 should capture exactly the same signal compared to the first channel in frame 2. Because the time difference may varies, so this procedure must be done in real-time.

|  |
| --- |
|  |
| 1. The need for real-time calibration for multiple Tx using TDM-MIMO. |

The implementation is done in pixel level and is as follows:

| postprocessing.extract\_tgt |
| --- |
| 1. class Tgt: 2. … 3. def find\_point\_doa(): 4. try: 5. tx\_ratio = point\_all\_channel[0] / point\_all\_channel[7] 6. except: 7. tx\_ratio = 1 8. point\_all\_channel[:4] = point\_all\_channel[:4] / tx\_ratio |

## Data collection

Raw data is collected to build samples for training. In this case, both radar ADC data and camera images are recorded simultaneously. Multi-threading is used to run the data capture and file writing at the same time:

.\mimo2go\_data\_collection\record\_radar\_and\_cam.py

A recording scheduler enables the data collection according to a planed schedule:

| mimo2go\_data\_collection.mimo2go\_record |
| --- |
| 1. from general\_tools.record\_auto\_schd import … |
| general\_tools.record\_auto\_schd |
| 1. def load\_auto\_record\_schd: 2. record\_hour = […] 3. record\_min = […] |

It means that for each of the minutes in record\_min in the correct hour, the radar will start recording for a configured time period:

| configuration.cfg\_main |
| --- |
| 1. class RadarConfigure: 2. def config\_recording(): 3. self.filetime = 60 |

The recorded radar data are located under the below folder in .hdf format

.\mimo2go\_data\_collection\data\_collected\date\hour\radar

To understand of the .hdf format, a software HDFView can be used. In this case, four channels are recorded: t1\_r1/2/3/4\_i frame by frame. Each frame has 258 chirps with 64 samples in one chirp. At the same time, timestamps at different frames are also recorded.

|  |
| --- |
|  |
| 1. A glance at recorded radar data files. |

And the video data are located under

.\mimo2go\_data\_collection\data\_collected\date\hour\video

The video data includes the video itself, and the timestamp in .hdf format. In this case to lighten the processor burden, instead of all frames, the timestamps of part of the frames are recorded. The recorded frame numbers are stored in /frames, and the timestamps are the time difference of each recorded frame compared to the start time. To read back the full timestamps, functions are called as follows:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics: 4. … 5. if self.use\_video: 6. … 7. video\_ts = self.data\_loader.get\_timestamp\_from\_video\_ts() |
| general\_tools.h5\_data |
| 1. class DataProcessor: 2. … 3. def get\_timestamp\_from\_video\_ts(): 4. … 5. ts\_list = self.read\_video\_timestamp() 6. def read\_video\_timestamp(): 7. … |
|  |
| 1. A glance at recorded video timestamp file. |

When the data collection is done on RPi, it is suggested to use the following scripts:

$ screen

$ python3 record\_radar\_and\_cam.py

## Labelling

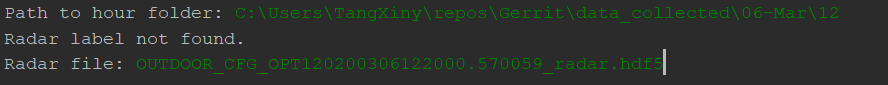
Once the data is collected, a labelling tool is developed for labelling purpose:

.\radar\_label\_sample\label\_tool\_v0.1.py

The labelling tool is modified from an open source labelling tool from computer vision. Before using it, make sure there is a video folder and a corresponding radar folder are properly located in a base folder:



In the above example, the base folder is .\xx\06-Mar\12\, and the two folders are located respectively. Within the two folders, the radar data (.hdf5) and video data (.avi for video and .hdf5 for timestamp) in the previous data collection task are stored. When the labelling tool is running, the user needs to input the base folder (not the radar folder), and then input the radar filename.



For the first time usage, there is no existing label file for this specific radar data and it reports “Radar label not found.” At the same time, a subfolder .\label\ is created, and a label file will be created under this folder with the same timestamp name as the chosen radar file. If there is already a label file available for the corresponding radar file, for each of the frame, the label information will be read into the tool first, and modification can be done accordingly:



When the radar file is chosen, the .hdf5 file shown in Figure 38 is read into the tool. As can be found, all channels are recorded and only one channel is used for labelling:

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. channel = “t1\_r1\_i” |

Other channels can be chosen if a better SNR is found.

Depends on the size of radar file, sometimes only part of the data is of interest, this part of interest is used to build “records” based on the frames’ timestamps. The ADC data is stored in image\_list, the frame number is stored in record\_frame\_list, the timestamp of each frame is stored in tss\_list, and the labels in each frame is stored in label\_list. If there is existing label files, these labels are first read and displayed. Then they can be modified if necessary. In the labelling tool, the records include all data within one hour as in build\_records function.

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. records = build\_records() 2. … 3. image\_list = records[0][“files”] 4. record\_frame\_list = records[0][“frames”] 5. tss\_list = records[0][“tss”] 6. label\_list = records[0][“labels”] 7. … 8. def build\_records(times): 9. … 10. end\_time = times[0] + timedelta(hours=1) 11. … |

For each of the frame, the ADC in certain channel is found from image\_list, and the corresponding timestamp from the radar can also be found with the same list index. Based on the radar timestamp, the video image is obtained by searching the closest timestamp from video.

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. radar\_ts = tss\_list[img\_index] 2. closest\_video\_frame = min(video\_ts, key=lambda x: abs(x-radar\_ts)) 3. video\_index = video\_ts.index(closest\_video\_frame) 4. … |

Now for each of the radar frame, the ADC can be used to generate RDM and the corresponding video frame is also available as shown in Figure 40. In this figure, a pedestrian on the left side is captured by both the camera and radar. With the labelling tool, the bouding box of the pedestrian in the radar image is produced. There are two options to produce the bounding box.

In option 1, the bounding box is produced by two mouse clicks, the first mouse click is to select the left up corner of the bounding box and the second mouse click is to select the right bottom corner of the bounding box. The example is shown in Figure 41.

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. … 2. point\_1 = (-1, -1) 3. bb\_rule = False 4. … 5. def mouse\_listener(): 6. elif event == cv2.EVENT\_LBUTTONDOWN: 7. … 8. else: 9. … 10. if point\_1[0] is -1: 11. … 12. point\_1 = (x, y) 13. else: 14. … 15. point\_2 = (x, y) |

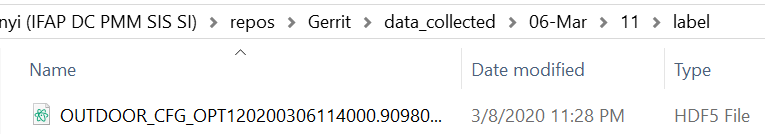
In option 2, the bounding box is produced by a single click of mouse, then the bounding box is grew according to a defined rule. The same rule will be used for inference. The example is shown in Figure 42.

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. … 2. point\_1 = (-1, -1) 3. bb\_rule = True 4. … 5. def mouse\_listener(): 6. elif event == cv2.EVENT\_LBUTTONDOWN: 7. if bb\_rule: 8. … 9. cfar\_prop\_bb = cfarProcessor.prop\_bb2() 10. … 11. point\_1 = (x\_1, y\_1) 12. point\_2 = (x\_2, y\_2) |
| detectors.get\_bb\_cfar |
| 1. def prop\_bb2(): 2. … 3. bbs = [] 4. for pnt in pnts: 5. … 6. bbs.append([self.find\_bb0/1()]) 7. … 8. def find\_bb0/1(): 9. … |

Keyboards are used for different actions such as changing frames and changing classes.

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. … 2. while True: 3. … 4. pressed\_key = cv2.waitKey(50) 5. if pressed\_key == ord(“a”): 6. img\_index = decrease\_index() 7. if pressed\_key == ord(“d”): 8. img\_index = increase\_index() 9. if pressed\_key == ord(“s”): 10. class\_index = decrease\_index() 11. if pressed\_key == ord(“w”): 12. class\_index = increase\_index() 13. if pressed\_key == ord(“c”): 14. records[0][“labels”][img\_index] = [] 15. if pressed\_key == ord(“q”): 16. save(records) |

When the records are saved, a label file with the same timestamp with the radar file in filename is generated, and these generated files are used for training.



The stored labels follows the following format:

|  |
| --- |
| radar\_label\_example.label\_tool\_v0.1 |
| 1. def yolo\_format(): 2. … 3. return [class\_index, x\_center, y\_center, x\_width, y\_height] 4. line = yolo\_format() 5. line.insert(0, real\_rdm\_index) 6. labels.append(np.array(line)) |

Note all position information are normalized to the picture size (in pixel). A glance of the .hdf5 label file is shown in Figure 43. When there are multiple targets (say N) in the same frame, the dimension size will be N x 6.

|  |
| --- |
|  |
| 1. Labelling tool which synchronizes the radar RDM and video image. |
|  |
| 1. Labelling option 1: produce bounding box by two mouse clicks. |

|  |
| --- |
|  |
| 1. Labelling option 2: produce bounding box by two mouse clicks. |

|  |
| --- |
|  |
| 1. A glance of the labeled .hdf5 file. |

## Demo video with classifiers

A model can be obtained by training of the above mentioned samples. In release 0.1, an rcnn model and an xgboost model are covered. It is possible to generate offline demo videos to observe the classification step by step with

.\demo\make\_demo\_pics.py

In realease 0.1, there are a few demo options, and viewing of raw images, images with labels for labelled data, and images with classified results are available to choose. There are two major parameters to set:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. if \_\_name\_\_ == “\_\_main\_\_”: 2. DEMO\_OPT = 0 or 1 or 2 3. WORK\_OPT = 0 or 1 or 2 |

* DEMO\_OPT = 0

The video includes three parts: left, center and right. Video image, single channel RDM and beamformed RDM are shown respectively. Single channel has the largest FoV from the antenna and the beamformed channel has a narrower FoV. There are different WORK\_OPT can be set.

* + WORK\_OPT = 0

Show RDMs without any bounding box. The purpose is to visualize the RDM of single channel and combined beam from multiple channels. In principle, the beamformed RDM has a higher SNR. An example is shown in Figure 44.

* + WORK\_OPT = 1

Show RDMs with labelled data (with bounding boxes and classes). The purpose is to verify all labels from the labelling tool. An example is shown in Figure 45.

* + WORK\_OPT = 2

Show RDMs with detections and classifications (with bounding boxes and classes). The purpose is to test the trained AI models offline. An example is shown in Figure 46.

|  |
| --- |
|  |
| 1. DEMO\_OPT = 0, WORK\_OPT = 0 |
|  |
| 1. DEMO\_OPT = 0, WORK\_OPT = 1 |
|  |
| 1. DEMO\_OPT = 0, WORK\_OPT = 2 |

* DEMO\_OPT = 1

The video includes four parts: left, center left, center right and right. Video image, RDMs of three different beams looking at left, center and right regions of the radar are shown respectively. It is clearly observed that the target in the correct region has the highest SNR and is most likely to be detected. There are also three WORK\_OPT to be set, and the purposes of each WORK\_OPT are the same as the description above.

The examples are shown in Figures 47 to 49.

|  |
| --- |
|  |
| 1. DEMO\_OPT = 1, WORK\_OPT = 0 |
|  |
| 1. DEMO\_OPT = 1, WORK\_OPT = 1 |
|  |
| 1. DEMO\_OPT = 1, WORK\_OPT = 2 |

* DEMO\_OPT = 2

The video includes three parts: left, center and right. Video image, single channel RDM, and positioned targets with postprocessing including tracking are shown respectively. In the right side position map, the radar is located at the center bottom, a tree is shown as a dot in front of the radar, other than that, lanes and gates are shown as background. Each grid represents for 5 meters, so the tree is roughly 3 meters away and the gates are roughly 20 meters away. There are two options to set for WORK\_OPT, which are WORK\_OPT = 1 and WORK\_OPT = 2. The reason is that to have the position of the targets, a bounding box is needed. Bounding boxes are only available when WORK\_OPT = 1 (labels) and WORK\_OPT = 2 (detections).

The examples are shown in Figure 50 and Figure 51.

|  |
| --- |
|  |
| 1. DEMO\_OPT = 2, WORK\_OPT = 1 |
|  |
| 1. DEMO\_OPT = 2, WORK\_OPT = 2 |

The purpose of all these DEMO\_OPT and WORK\_OPT options is to monitor the every phase of development start from raw data to detection and to classification, from single channel data to combined RDM at different regions with higher SNR.

After selecting the option combination, the radar configuration file and data folder are loaded, note in the data folder, there are a few subfolders:

.\[input folder]\radar\

.\[input folder]\label\ (for WORK\_OPT = 1)

.\[input folder]\video\

As stated in Chapter 4, for different DEMO\_OPT and WORK\_OPT, setups are done in

|  |
| --- |
| demo.make\_demo\_pics |
| 1. if \_\_name\_\_ == “\_\_main\_\_”: 2. … 3. demo = DemoOffline() 4. … 5. class DemoOffline: 6. def \_\_init\_\_(): 7. … |

It is important to note that for MIMO systems, the recorded data includes a few single channel data, and these data could be combined into different beams to obtain higher SNR for certain regions. However, the data that is used for inference is just one RDM, which can be from one channel or one beam. Even for single channel situation, the SNR of different channels can be different, e.g. the RX antenna with the closest distance with the TX antenna may be relatively noisy, or some antenna may experience higher reflection from the radome.

The choice of which single channel to be used is in

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def demo\_opt(): 4. self.single\_rdm = True 5. self.single\_rdm\_ch = [0] |

Beamed channels are built with calibrated single channels. The calibration matrix is obtained from Chapter 6.1 and is set in configuration file as stated in Chapter 2.3.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def cvt\_adc\_data\_to\_list(): 4. … 5. return radar\_raw\_cali |

The beam is looking at regions with different angles, a corresponding vector is generated and the beam is obtained when these calibrated channels multiply this vector. The looking angles are set in code line 330.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def get\_beam\_vec\_1d(): 4. beam\_angles = np.array([-17, 0, 17]) \* np.pi / 180 5. … 6. return vecs 7. def make\_demo\_pics(): 8. … 9. if self.beamed\_rdm: 10. … 11. for ich in range(len(vec)): 12. adc\_multi[ich] = … 13. … |

Similar to single channel data, when the beamed channels are obtained, user will need to decide which one(s) should be used:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def demo\_opt(): 4. self.beamed\_rdm = True 5. self.beamed\_rdm\_ch = [0, 1, 2] |

Note the selected channel corresponds to code line 330. There is also a possibility to feed the beamed data to AI processors when the self.single\_rdm is False:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. if self.beamed\_rdm: 6. … 7. if not self.single\_rdm: 8. rcnn\_channel = rdm\_beams[0] |

Other important setups include the setups for classification and tracking:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. if self.enable\_det\_class: 6. self.cls\_score\_thresh = 0.6 7. self.iou\_thresh = 0.5 8. if self.enable\_tracker: 9. self.mytracker = tracker() 10. self.frame\_period = 0.12 |

In code line 362, the interframe period is defined to predict the current location of the targets based on the previous measurement on location and speed.

When all these setups are finished, the radar data is loaded to create “records” according to the timestamps.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. self.records = self.build\_records() 6. def build\_records(): 7. … 8. insert\_records2(records, “2020,3,4,11,25,10”, “2020,3,4,11,25,21”) 9. insert\_records2(…) |

At code line 370, the start timestamp and the ending timestamp are added in this format. When the main code is running, it is required to input the base folder as stated before code line 310, all the radar data is loaded into the program, but only the data with timestamps in between of the two input timestamps are added into the records, and the demo will be generated from the records only. There can be multiple input of the records as at code line 371. Within the built records, there are a few keys: “files” is used to contain the multi-channel radar raw adc data of the frame; “labels” contains the labels if there is any for the frame, if there is no labelled bounding box in the frame, in list is empty for this frame; “tss” contains the timestamps of each frames and “frames” contain the frame number at each file. Make sure that the start and ending timestamps in each line are contained in the same data file, in this case, the “frames” does not contain repeated frame numbers. If the video file is available, the closest video frame will be obtained from the radar timestamp information.

For each of the frame in the records, the adc data for all channels is found and calibrated:

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. raw\_data\_full = r[“files”][iframe] 6. adc\_data\_full = self.cvt\_adc\_data\_to\_list(raw\_data\_full) 7. rdms\_all\_data = process\_full\_channels(adc\_data\_full) |

It shows that rdms from all channels are based on calibrated results. Note the computed rdms are for DoA estimation, simple 2dFFT was used without any other filters.

Then one of the channel is selected for AI processor input, here filters can be applied as requested.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. if self.single\_rdm: 6. adc\_data\_rcnn = adc\_data\_full[self.single\_rdm\_ch[0]] 7. rdms\_single\_rcnn = process\_rcnn\_channels(adc\_data\_rcnn) 8. rcnn\_channel = rdm\_single\_rcnn 9. if self.beamed\_rdm: 10. adc\_multi = np.copy(adc\_data\_full) 11. adc\_multi = adc\_multi\*vec 12. rdm\_beams.append(process\_rcnn\_channels(adc\_multi)) 13. if not self.single\_rdm: 14. rcnn\_channel = rdm\_beams[0] |

The bounding boxes and classes are produced by either the label files (WORK\_OPT = 1) or the detectors and classifiers (WORK\_OPT = 2). For both cases, the list labels is updated for each frame.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. labels = [] 6. if self.use\_labels: 7. labels = r[“labels”][iframe] 8. else: 9. if self.detector\_type == “rcnn”: 10. labels = ml\_framework.get\_labels(rcnn\_channel) 11. if self.detector\_type == “cfar”: 12. bbs = detector.get\_bb(rcnn\_channel) 13. labels = ml\_framework.get\_labels(rcnn\_channel, bbs) |

Once the labels are obtained, and DEMO\_OPT = 2, post processings are applied as stated in Chapter 5. Otherwise, if DEMO\_OPT = 1, the bounding boxes and RDM are plotted.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. if self.target\_pos: 6. tgts = extract\_tgt(rdms\_all\_channel, labels) 7. if self.enable\_tracker: 8. tgts =apply\_tracker(tgts, self.frame\_period) |

From Chapter 5.2, for a given target, if it is successfully tracked in the current frame, the tracking level will increase by one. Otherwise the tracking level will be reduced by one. In the visualization or other attempts, this tracking level can be used for filtering. Because the tracking is based on positions, it only work when DEMO\_OPT = 2. Note additional DoA offset should be applied due to the installation. Here, when the tracking level is larger than or equal to 1, the result will be sent (displayed in this case).

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def make\_demo\_pics(): 4. … 5. if self.target\_pos: 6. viz.cv2\_show\_rdm(tgts, save=1, doa\_post\_calibration=cfg.radar\_cfg.azimuth\_offset\_deg) |
| postprocessing.viz |
| 1. def cv2\_show\_xy: 2. for tgt in tgts: 3. if tgt[3] >= 1: 4. azimuth\_doa\_offset() |

### Stand-alone demo video without classifier

The stand-alone module handles the situation where no classification is needed. It will detect targets from RDM for each frame with bounding boxes, then the target information from the bounding boxes will be extracted and tracked as stated above. Labels are still produced, but with a class value of -1.

|  |
| --- |
| demo.make\_demo\_pics |
| 1. class DemoOffline: 2. … 3. def work\_opt(): 4. self.load\_detector(det\_type=”cfar”) 5. self.load\_classifier(cls\_type=”non”) 6. def make\_demo\_pics(): 7. … 8. if self.classifier\_type == “non”: 9. for bb in bbs: 10. labels.append([0, -1, bb]) |

References

1. AN553 *Position2Go-24GHz radar kit with BGT24MTR12 and XMC4700 32-bit ARM® CortexTM-M4 MCU for ranging and target position estimation.*
2. A. V. Oppenheim, A. S. Willsky, and S. Hamid Nawab, Signals and systems, 2nd Edition.
3. B. R. Mahafza, Radar systems analysis and design using MATLAB®, 2nd Edition.
4. B. Lindmark, “Comparison of mutual coupling compensation to dummy columns in adaptive antenna systems”, IEEE Trans. Antenna Propag., vol. 53, no. 4, pp. 1332-1336, 2005.

Revision history

| Document version | Date of release | Description of changes |
| --- | --- | --- |
| 0.1 | 2020.3.16 | @ Tang Xinyi (IFAP DC PMM SIS SI) |
|  |  |  |
|  |  |  |