Handbook

User Guide

Handbook

User Guide

Author: Acconeer AB

Version:a121-v1.8.1

Acconeer AB November 8, 2024

Contents

1 Acconeer SDK Documentation Overview 5

I Pulsed Coherent Radar 6

2 Radar principles 6
2.1 Reflectivity . . . 6
2.2 Radar cross section . . . 6

3 Physical integration 7

II A121 8

4 Introduction 9

5 Differences between A121 and A111 9

6 Measurement range 9
6.1 Configuration . . . 10
6.2 Limitations . . . 10
6.3 Caveats . . . 11

7 Frames, sweeps and subsweeps 11
7.1 Subsweeps . . . 12
7.2 Measurement execution order . . . 12
7.3 Limitations . . . 13

8 Interpreting radar data 13

9 Profiles 15

10 Signal strength 16

11 How to configure 16
11.1 Considerations . . . 16
11.2 Range-related parameters . . . 16
11.3 Rate-related parameters . . . 17
11.4 Idle-related parameters and control . . . 18
11.5 Other parameters . . . 18

12 Timing 18
12.1 Notation . . . 18
12.2 Overview . . . 19
12.3 Sample duration . . . 19
12.4 Point duration . . . 19
12.5 Subsweep duration . . . 19
12.6 Sweep duration . . . 20
12.7 Sweep period . . . 20
12.8 Frame duration . . . 20
12.9 Frame period . . . 20
12.10 Fixed overheads and high speed mode (HSM) . . . 21

13 Detectors 21
13.1 Distance detector . . . 21
13.2 Presence detector . . . 21

14 In-depth topics 21
14.1 Figure of Merits . . . 21
14.2 Continuous sweep mode (CSM) . . . 23
14.3 Loopback . . . 24

14.4 Control 27

III A111 28

15 Profiles 29

16 Typical ranges for different objects 31

17 Configuration overview 32
17.1 Signal averaging and gain 32
17.2 Sweep and update rate 32
17.3 Repetition modes 32
17.4 Power save modes 33
17.5 Configuration summary 33

18 Services and detectors overview 33

19 Services 34
19.1 Power Bins 34
19.2 Envelope 39
19.3 IQ 45
19.4 Sparse 50

20 Detectors 58
20.1 Distance detector 59
20.2 Presence detector 59
20.3 Obstacle detector 59

21 System overview 59

22 Performance metrics 60
22.1 Radar loop gain 61
22.2 Depth resolution 61
22.3 Distance resolution 61
22.4 Half-power beamwidth 61
22.5 Distance jitter 61
22.6 Distance linearity 61
22.7 Update rate accuracy 62
22.8 Close-in range 62
22.9 Power consumption 62

IV Appendix 63

23 Disclaimer 63

1 Acconeer SDK Documentation Overview

To better understand what SDK document to use, a summary of the documents are shown in the table below.

Table 1: SDK document overview.

Name	Description	When to use
	RSS API documentation (html)	
rss_api	The complete C API documentation.	- RSS application implementation - Understanding RSS API functions
	User guides (PDF)	
A121 Assembly Test	Describes the Acconeer assembly test functionality.	- Bring-up of HW/SW - Production test implementation
A121 Breathing Reference Application	Describes the functionality of the Breathing Reference Application.	- Working with the Breathing Reference Application
A121 Distance Detector	Describes usage and algorithms of the Distance Detector.	- Working with the Distance Detector
A121 SW Integration	Describes how to implement each integration function needed to use the Acconeer sensor.	- SW implementation of custom HW integration
A121 Presence Detector	Describes usage and algorithms of the Presence Detector.	- Working with the Presence Detector
A121 Smart Presence Reference Application	Describes the functionality of the Smart Presence Reference Application.	- Working with the Smart Presence Reference Application
A121 Sparse IQ Service	Describes usage of the Sparse IQ Service.	- Working with the Sparse IQ Service
A121 Tank Level Reference Application	Describes the functionality of the Tank Level Reference Application.	- Working with the Tank Level Reference Application
A121 Touchless Button Reference Application	Describes the functionality of the Touchless Button Reference Application.	- Working with the Touchless Button Reference Application
A121 Parking Reference Application	Describes the functionality of the Parking Reference Application.	- Working with the Parking Reference Application
A121 STM32CubeIDE	Describes the flow of taking an Acconeer SDK and integrate into STM32CubeIDE.	- Using STM32CubeIDE
A121 Raspberry Pi Software	Describes how to develop for Raspberry Pi.	- Working with Raspberry Pi
A121 Ripple	Describes how to develop for Ripple.	- Working with Ripple on Raspberry Pi
XM125 Software	Describes how to develop for XM125.	- Working with XM125
XM126 Software	Describes how to develop for XM126.	- Working with XM126
I2C Distance Detector	Describes the functionality of the I2C Distance Detector Application.	- Working with the I2C Distance Detector Application
I2C Presence Detector	Describes the functionality of the I2C Presence Detector Application.	- Working with the I2C Presence Detector Application
I2C Breathing Reference Application	Describes the functionality of the I2C Breathing Reference Application.	- Working with the I2C Breathing Reference Application
	Handbook (PDF)	
Handbook	Describes different aspects of the Acconeer offer, for example radar principles and how to configure	- To understand the Acconeer sensor - Use case evaluation
	Readme (txt)	
README	Various target specific information and links	- After SDK download

Part I

Pulsed Coherent Radar

2 Radar principles

2.1 Reflectivity

The amount of energy received back to the Rx antenna depends on the reflectivity of the object (γ), the radar cross section (RCS) of the object (σ), and the distance to the object (R). A reflection occurs when there is a difference in relative permittivity between two media that the signal is propagating through. γ is then given as

$$\gamma = \left(\frac{\sqrt{\varepsilon_1} - \sqrt{\varepsilon_2}}{\sqrt{\varepsilon_1} + \sqrt{\varepsilon_2}} \right)^2 \tag{1}$$

where ε_1 and ε_2 is the relative permittivity, at 60 GHz, on either side of the boundary. Keep in mind that the relative permittivity is generally frequency dependent and may also vary depending on the exact material composiion and manufacturing process. Table 2 lists approximate values for the real part of the relative permittivity for some common materials.

Table 2: Approximate relative permittivity of common materials

Material	Re(ε) at 60 GHz	γ with air boundary
Air	1	0
ABS	2.5-4.0	0.05 - 0.11
Polyethylene (PE)	2.3	0.042
Polypropylene (PP)	2.2	0.038
Polycarbonate	2.75	0.06
Mobile phone glass	6.9	0.2
Plaster	2.7	0.059
Concrete	4	0.11
Wood	2.4	0.046
Textile	2	0.029
Metal	–	1
Human skin	8	0.22
Water	11.1	0.28

Table 2 shows that some materials are semi-transparent to 60 GHz signals and it is hence possible to detect reflecting objects behind a surface of these materials, each boundary with a change in permittivity gives a reflection. This is a useful property in applications where the sensor measures through the product housing or when detecting objects behind other objects such as walls and clothing. For optimal design of the product housing, refer to the radome chapter in the “Hardware and physical integration guideline” document.

2.2 Radar cross section

The radar cross section is the effective area of the object that the signal is reflected against, for simple geometrical shapes, where the size is larger than the wavelength of the signal (~5 mm) and is in the far-field distance, it can be expressed analytically as in Figure 1. The far-field distance depends on the object size and its distance to the radar source. Generally speaking, far-field applies when the waves reflected by the object can be considered plane-waves. Representative back scattering pattern of a sphere, flat plate and trihedral corner reflector are shown in the polar plots. It is seen that the objects can have different maximum RCS, but also different radiation patterns, a flat plate for instance is very directive and if tilted away from the radar, the received energy will be decreased, whereas the corner has less angular dependence and is a more robust reflector in terms of angle with respect to the radar.

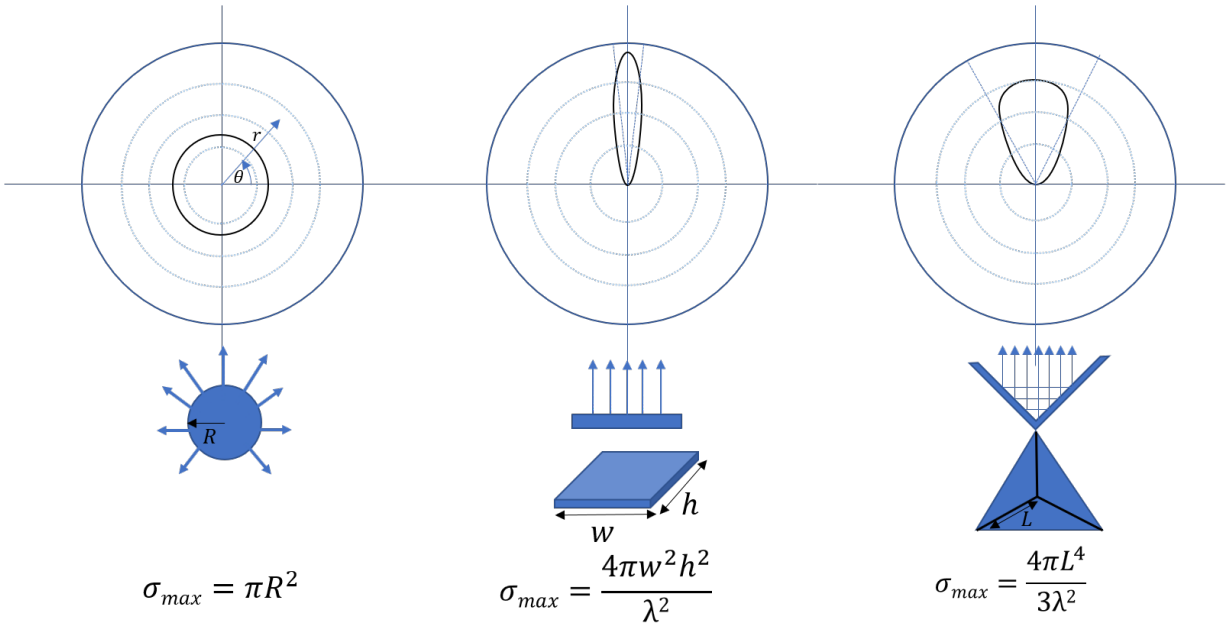


Figure 1: Radiation pattern and analytical expressions for simple geometrical shapes.

For most objects it is not possible to analytically calculate σ, instead it needs to be measured or modelled.

3 Physical integration

The A111 sensor contains the mmWave front-end, digital control logic, digitization of received signal and memory, all in one package. To integrate it in your application it is required to have a reference frequency or XTAL (24 MHz), 1.8 V supply, and a host processor, as illustrated in Figure 2, supported platforms and reference schematics are available at developer.acconeer.com.

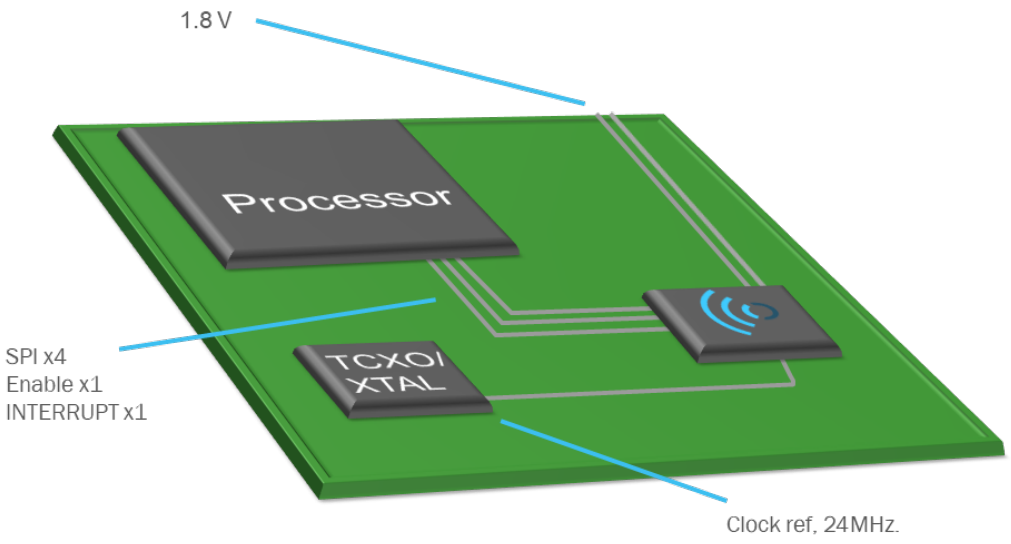


Figure 2: Illustration of integration into host platform, the A111 is marked with the Acconeer logo.

In addition to the above it is also important for optimized integration to consider the electromagnetic (EM) environment, both in terms of what is placed on top of the sensor as well as to the side of the sensor. To evaluate the EM integration a Radar loop measurement can be conducted by placing an object in front of the sensor and rotating the sensor around its own axis, as illustrated in Figure 3. The received energy from e.g. the Envelope Service can then be used to plot the amplitude versus rotation angle (θ).

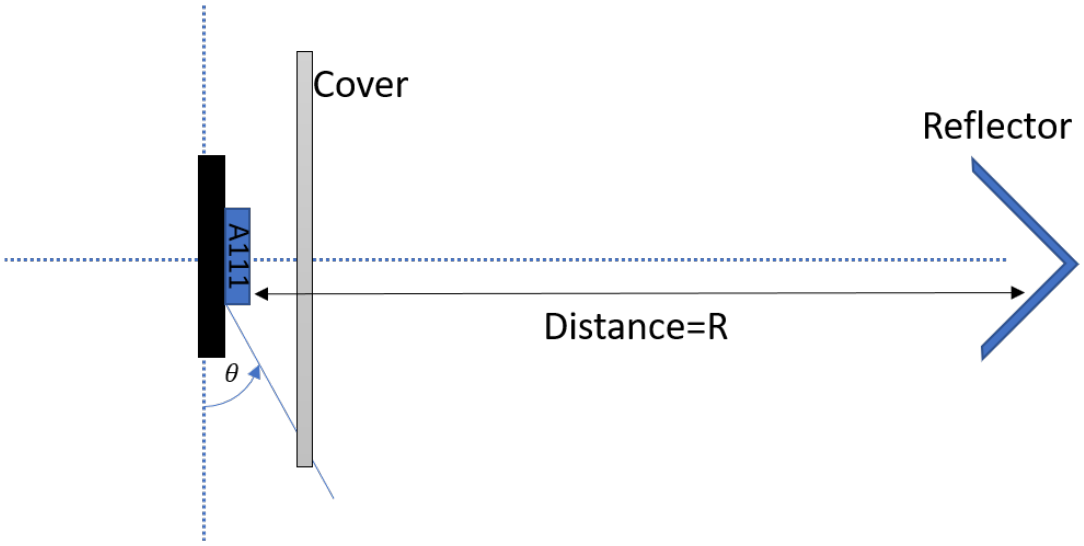


Figure 3: Setup configuration for radar loop pattern measurements.

The radiation pattern of the integrated antennas will be affected by anything that is put on top of the sensor as a cover. The transmission through a material is given by 1-γ, where γ is the reflectivity calculated in Equation 3. Hence, materials with low reflectivity are good materials to use as a cover on top of the sensor, plastic is a good choice and the sensor is not sensitive to the color of the material. Figure 4 shows the measured Radar loop pattern for 3 different scenarios, plastic (ABS), gorilla glass (GorillaGlass) and free space (FS). To further optimize the cover integration the thickness of the material should be considered. One can also use a layered cover which uses materials of different ε for optimum matching to the medium in which the signal is going to propagate or even to increase the directivity, as shown in Figure 4, where the beam width has been decreased by adding material on top of the sensor. More information on the EM integration aspects can be found in the "Hardware and physical integration guideline" document available at developer.acconeer.com.

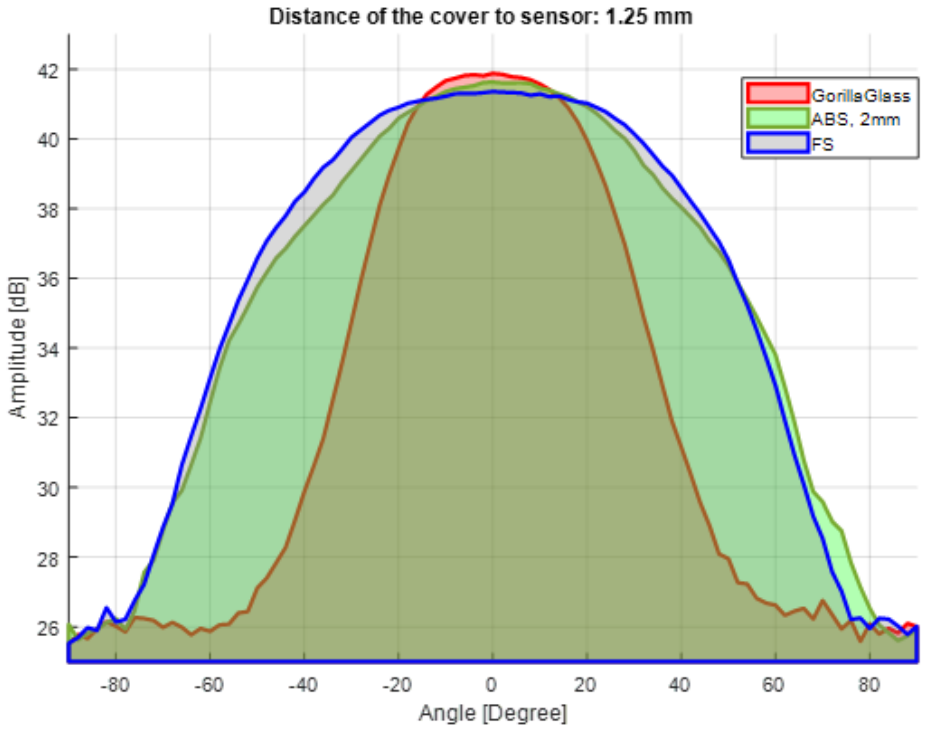


Figure 4: Integration of sensor cover and how different materials impact the radiation pattern on the H-plane. The object used is a trihedral corner of radius 5 cm.

Part II

A121

4 Introduction

The *sparse IQ* service is a new way of utilizing the Acconeer pulsed coherent radar (PCR) technology. As the name implies, it produces complex (IQ) data points with essentially arbitrary distance spacing (sparse). This gives the flexibility needed to set up the system for maximum efficiency across a wide range of applications.

5 Differences between A121 and A111

If you're new to Acconeer products and haven't used the A111 before, skip this section.

Here are some of the main differences, additions, and highlights coming from A111:

- The four services available for the A111 – *power bins*, *envelope*, *IQ*, and *sparse* – are all superseded by the *sparse IQ* service. This new service combines the individual strengths of all A111 services into one, without any sacrifices or compromises.
- Range is now set in a discrete "point" scale instead of meters, which allows full control over the measured range. The range is no longer limited to 60 mm intervals.
- In the new range scale, the *downsampling factor* of A111 is replaced by *step length*. This also brings a much wider range of settings, starting at 2.5 mm.
- Any number of sweeps may be measured per frame, only limited by buffer size.
- The sensor buffer now holds 4095 complex points.
- The floating point gain scale is replaced by a much wider integer scale, allowing for more precise control and removing the need for *maximize signal attenuation*.
- *Power save mode* is replaced by *inter sweep/frame idle states*, meaning it's now possible to set the idle state between sweeps as well as between frames.
- *MUR* is replaced by *PRF* with similar behavior but a wider range of settings.
- The *repetition mode* is simplified to a *frame rate* setting. If set, it corresponds to using the *streaming* mode. If not set, it corresponds to using the *on demand* mode.
- The *get next* function is split up into four functions for more flexible control over the measurement execution flow – *measure*, *wait for interrupt*, *read*, and *execute processing*. This also removes the need for the *asynchronous measurement* setting.
- The sensor can now be fully reconfigured on the fly via the *prepare* function, which loads a (new) configuration onto the sensor.
- The sweep can now be split up into *subsweeps* with different configurations, for example allowing you to use different profiles across the range.

6 Measurement range

The sparse IQ service can be configured to measure practically any range of distances in a so-called *sweep*. In other words, a sweep makes up the points in space where reflected pulses are measured.

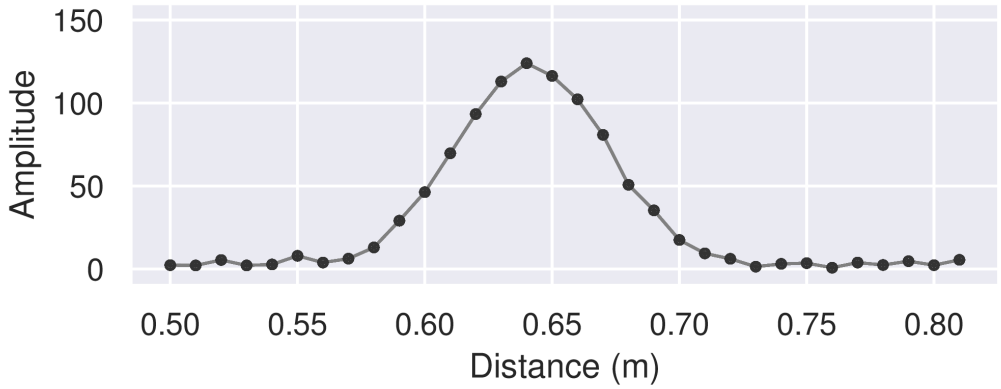


Figure 5: Amplitude of a mocked sweep of an environment with a single object.

Figure 5 above shows an example of a sweep with a range spanning from a start point at ~ 0.50 m to an end point at ~ 0.81 m. Here, a total of 32 points were measured with a distance between them of ~ 10 mm, also configurable. The shortest configurable distance between points is ~ 2.5 mm.

The more points that are measured, the more memory is used and the longer it takes to measure the sweep. This may in turn lead to higher overall duty cycle and power consumption. Thus, it is often important to try to minimize the number of points measured, typically achieved by maximizing the distance between the points.

6.1 Configuration

For preciseness, the range is configured in a discrete scale with three integer parameters – the *start point* d_1, the *number of points* N_d, and the *step length* Δd. The *step length* corresponds to the distance between the points (mentioned above). The distance between the points in the discrete scale is ~ 2.5 mm, which is also why the shortest configurable distance between points is just that.

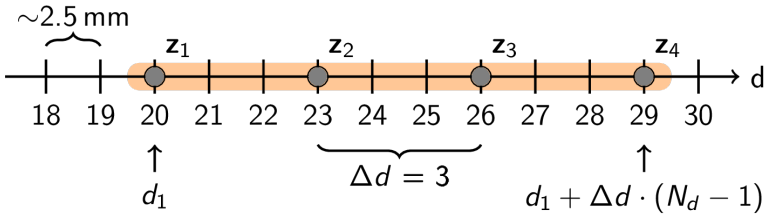


Figure 6: An illustration of the sweep *range* concept.

Figure 6 above demonstrates how a range can be set up with these parameters. The start point $d_1 = 20$, the number of points $N_d = 4$, and the step length $\Delta d = 3$ This gives the discrete points $\{20, 23, 26, 29\}$ which correspond to $\{50.0mm, 57.5mm, 65.0mm, 72.5mm\}$.

Note that the possible values for step length Δd are limited.

6.2 Limitations

The only limitation on the number of points N_d itself is related to the available buffer size of 4095 complex numbers. The buffer usage is the number of points N_d times the number of sweeps per frame N_s (see *Frames, sweeps and subsweeps*). In short, $N_d \cdot N_s \leq 4095$.

The step length must be a divisor or multiple of 24. The shortest step length, 1, gives a distance between points of ~ 2.5 mm. See Table 3 for an overview.

Table 3: Overview of selectable step lengths.

Step length		Step length cont'd	
Setting Δd	Distance	Setting Δd	Distance
1	2.5 mm	24	60 mm
2	5.0 mm	48	120 mm
3	7.5 mm	72	180 mm
4	10.0 mm	96	240 mm
6	15.0 mm	120	300 mm
8	20.0 mm	144	360 mm
12	30.0 mm	...	...

The *maximum measurable distance (MMD)*, i.e., the farthest configurable "end point", is limited by the *pulse repetition frequency (PRF)*. The lower PRF, the longer MMD.

The PRF also gives the *maximum unambiguous range (MUR)* – the maximum range at which target can be located at while still guaranteeing that the reflected pulse corresponds to the most recent transmitted pulse. Again, the lower PRF, the longer MUR.

See *Frames, sweeps and subsweeps* on the PRF for more details.

6.3 Caveats

A number of factors affect the actual real world distance of a given range point:

- The refractive index and thickness of materials the radar signal pass through.
- Systematic errors due to process, supply voltage, and temperature variations.
- Reference clock frequency.

Some static offsets can be compensated for by doing a *loopback* measurement of the "zero point".

7 Frames, sweeps and subsweeps

The sparse IQ service may be configured to perform several *sweeps* at a time in a so-called *frame*. Thus, every *frame* received from the sensor consists of a number of *sweeps*. Every sweep in turn consists of a number of *points* spanning the configured distance *range* (see *Measurement range*).

Some examples of suitable applications for multiple sweeps per frame are detecting motions, measuring velocities, and tracking objects.

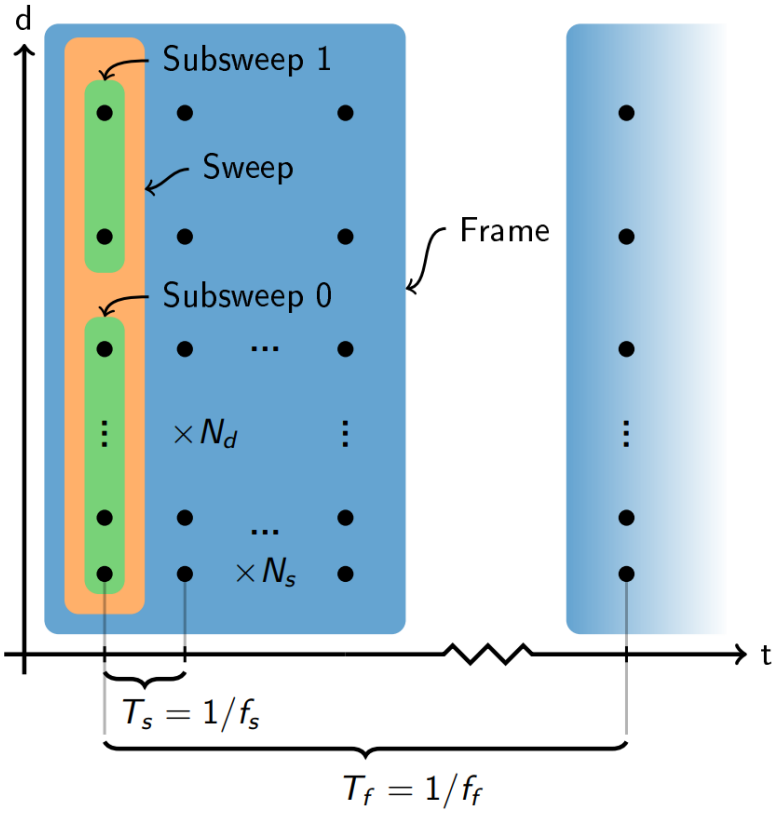


Figure 7: An illustration of the *sweep* and *frame* concept.

As shown in Figure 7, N_s is the number of *sweeps per frame (SPF)*. Typical values range from 1 to 64. The sweeps are sampled consecutively, where T_s is the time between two corresponding points in consecutive sweeps. This value is typically specified as the *sweep rate* $f_s = 1/T_s$. It is given by the sensor configuration, but is optionally **limited** to a fixed rate, letting the sensor idle between sweeps. Typical sweep rates range from 1 kHz to 10 kHz.

In a similar fashion as for sweeps, the *frame rate* is defined as $f_f = 1/T_f$. Typical values range from 1 Hz to 100 Hz. The sensor may idle in efficient low power states between frames, so maximizing the idle time between frames is crucial for minimizing the overall power consumption.

$$T_f \geq N_s \cdot T_s \Leftrightarrow f_f \leq f_s/N_s \tag{2}$$

The timing of frames can be done in two ways – either by letting the host trigger measurements of new frames, or by letting the sensor itself trigger on a periodic timer.

7.1 Subsweeps

The purpose of the subsweeps is to offer more flexibility when configuring the sparse IQ service. As the name implies, a subsweep represent a sub-region in the overall sweep. Each subsweep can be configured independently of other subsweeps, e.g. two subsweeps can be configured with overlapping range and different profiles.

The concept is utilized in the Distance detector, where the measured range is split into subsweeps, each configured with increasing HWAAS and Profile, to maintain SNR throughout the sweep, as the signal strength decrease with the distance.

7.2 Measurement execution order

As previously discussed, a frame consists of one or more sweeps, which in turn can be divided into multiple subsweeps. The execution order is as follows:

- The points are measured along the distances defined by *start point*, *num points* and *step length*.
- If subsweeps are utilized, they are measured in the order defined by the sensor configuration.
- After the measurement of the sweep (potentially containing subsweeps) is completed, the next sweep is measured.
- This is repeated until *sweeps per frame* sweeps have been measured

- Lastly the frame is formed by stacking the sweeps.

The following example illustrates the concept:

Assume a sensor configuration with three subsweeps and two sweeps per frame. Firstly, points are measured according to the distances specified by the first subsweep, followed by the measurements according to the second and third subsweep. Next, a second sweep is performed with the same subsweep configuration. Lastly, the two sweeps, containing three subsweeps, are stacked to form the frame.

7.3 Limitations

As with the number of points N_d, the only limitation on the number of sweeps per frame N_s itself is related to the available buffer size of 4095 complex numbers. The buffer usage is the number of points N_d times the number of sweeps per frame N_s. In short, $N_d \cdot N_s \leq 4095$.

8 Interpreting radar data

The data produced by the A121 sparse IQ service is conceptually similar to that of the A111 IQ service. The A121 Sparse IQ data is represented by complex numbers, one for each distance sampled. Each number has an *amplitude* and a *phase*, the amplitude is obtained by taking the absolute value of the complex number and the phase is obtained by taking the argument of the same complex number. A *sweep* is a array of these complex values corresponding to an amplitude and phase of the reflected pulses in the configured range.

For any given frame, we let $z(s,d)$ be the complex IQ value (point) for a sweep s and a distance point d.

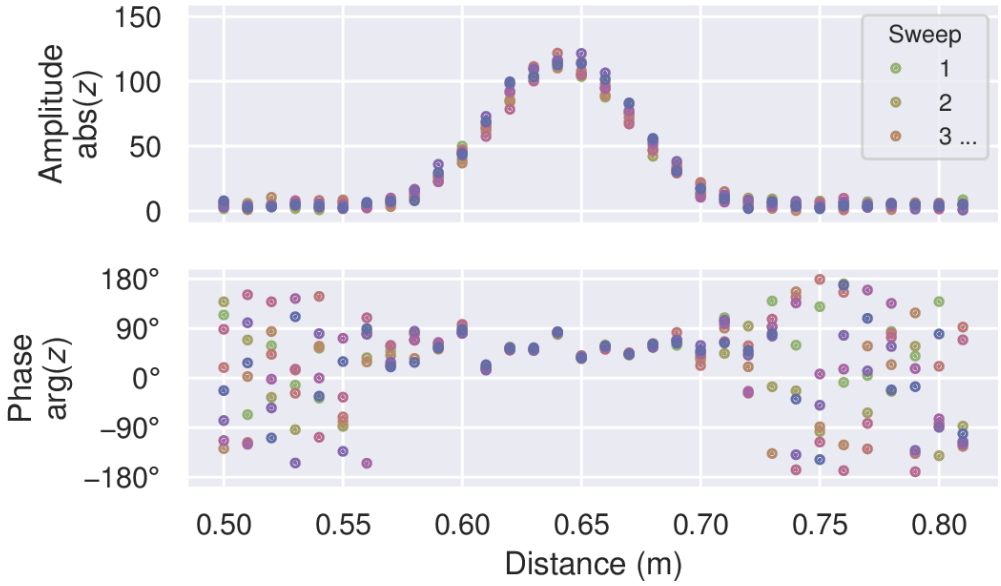


Figure 8: Mocked data of an environment with a single **static** object.

In the simplest case, we have a static environment in the sensor range. Figure 8 shows an example of this with a single static object at roughly 0.64 m. The sweeps in the frame have similar amplitude and phase, and the variations are due to random errors. As such, they could be *coherently* averaged together to linearly increase signal-to-noise ratio (SNR). In this context, coherently simply means “in the complex plane”.

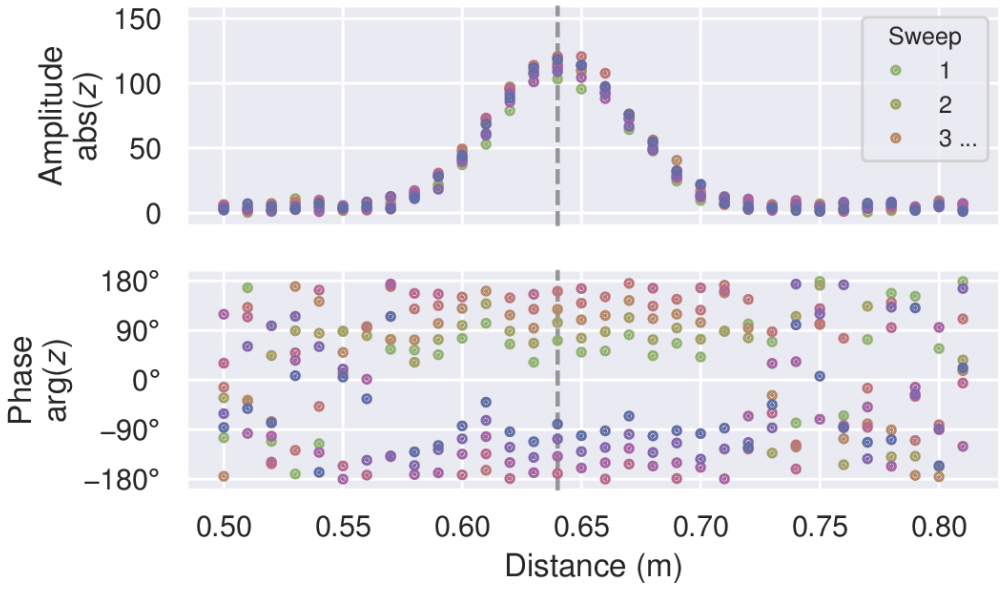


Figure 9: Mocked data of an environment with a single **moving** object.

In many cases, we want to track and/or detect moving objects in the range. This is demonstrated in Figure 9, where the object has moved during the measurement of the frame. The sweeps still have roughly the same amplitude, but the phase is changing. Due to this, we can no longer coherently average the sweeps together. However, we can still (noncoherently) average the amplitudes.

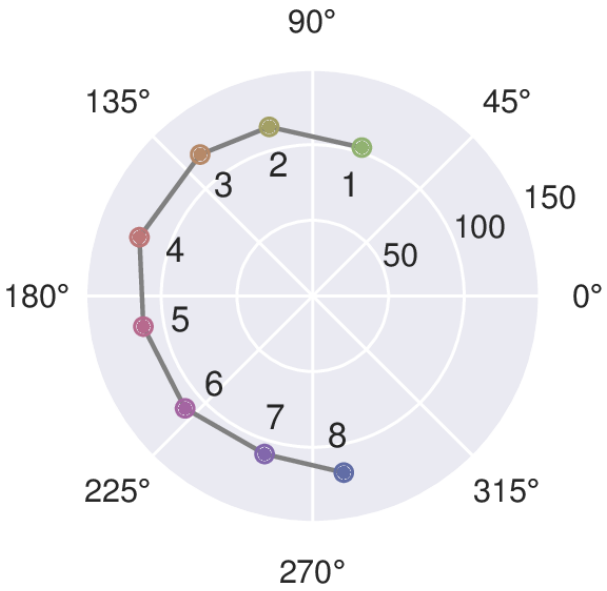


Figure 10: A slice of the mocked data in Figure 9 of an environment with a single moving object, shown in the complex plane.

To track objects over long distances we may track the amplitude peak as it moves, but for accurately measuring finer motions we need to look at the phase. Figure 10 shows the slice of the data along the dashed vertical line in Figure 9. Over the 8 sweeps in the example frame, the phase changed ~ 210°. A full phase rotation of 360° translates to $\lambda_{RF}/2 \approx 2.5$mm, so the 210° corresponds to ~ 1.5 mm.

As evident from the example above, even the smallest movements change the phase and thus move the signal in the complex plane. This is utilized in for example the *presence detector*, which can detect the presence of humans and animals from their breathing motion. It can also be used to detect a change in signal very close to the sensor, creating a "touchless button".

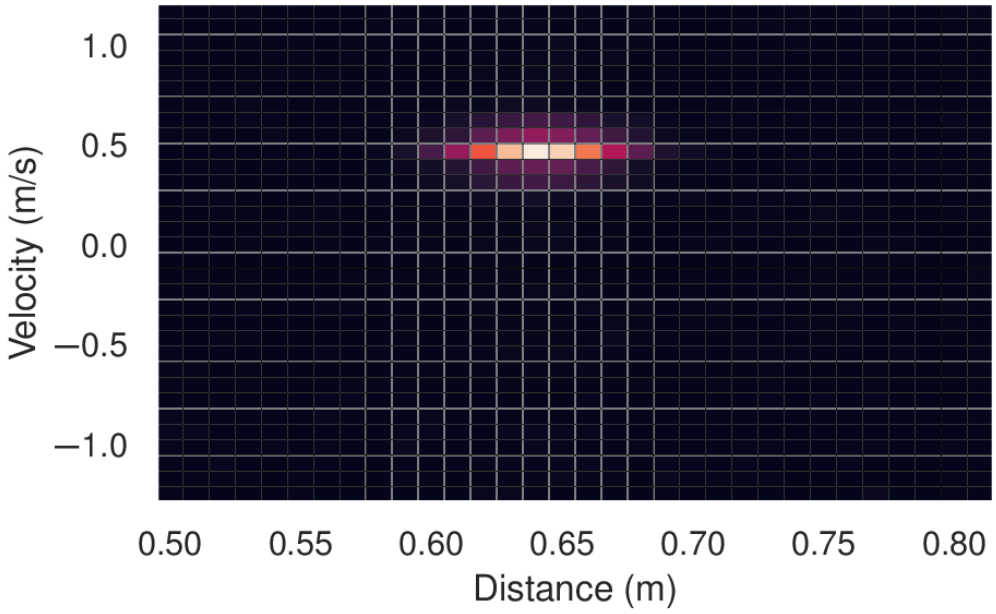


Figure 11: Mocked data of a single moving object, transformed into a distance-velocity (a.k.a. range-Doppler) map.

As shown, the complex data can be used to track the relative movement of an object. By combining this information with the sweep rate, we can also determine its velocity. In practice, this is commonly done by applying the fast Fourier transform (FFT) to the frame over sweeps, giving a distance-velocity (a.k.a.range-Doppler) map. Figure 11 shows an example of this in which we can see an object at ~ 0.64 m with a radial velocity of ~ 0.5 m/s (moving away from the sensor). This method is commonly used for applications such as micro and macro gesture recognition, velocity measurements, and object tracking.

9 Profiles

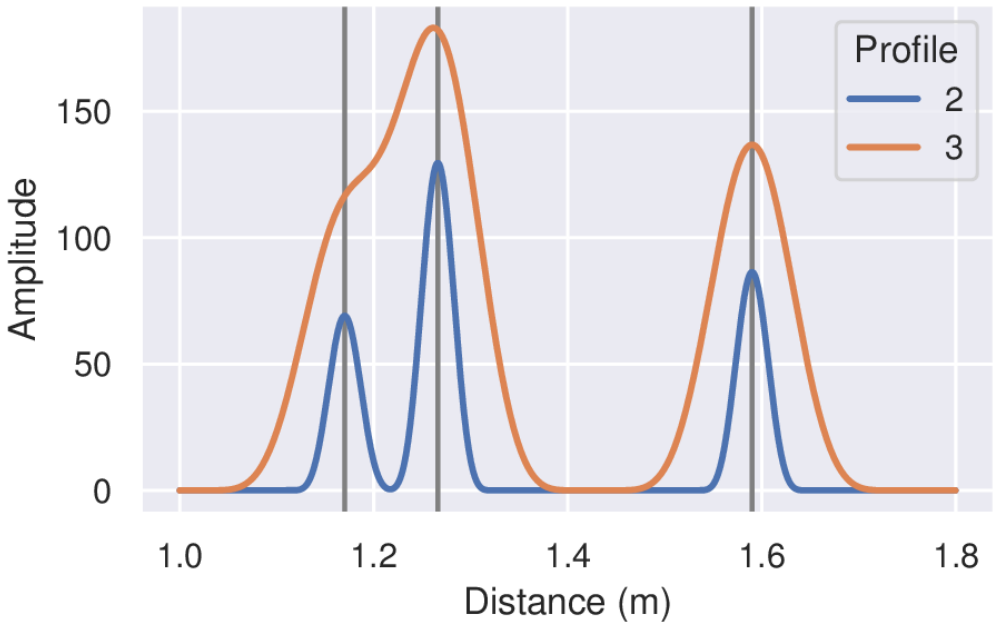


Figure 12: Mocked data for different profiles with three objects in the range.

One of the most important configuration parameters is the *profile*, which mainly sets the duration and shape of emitted pulses. Other internal parameters are set up accordingly to maximize the efficiency of the system, which affects the measurement time of a point. Higher numbered profiles use longer pulses, which generally:

- Increases SNR due to increased emitted energy.
- Decreases measurement time for a given configuration.
- Gives the possibility to sample more sparsely, decreasing measurement time and memory usage.

On the flip side, longer pulses also:

- Decreases precision due to lower bandwidth.
- Increases TX to RX leakage length, i.e., how far into the range the transmitted pulse is visible. The closest usable range due to this is referred to as the "close-in distance".
- Decreases distance resolution (ability to resolve objects close to each other).

Figure 12 illustrates the difference between two profiles. Profile 2 correctly resolves three objects where 3 cannot. However, profile 3 gives more energy for the same object.

10 Signal strength

To increase the signal-to-noise ratio (SNR), sampling of points may be repeated and averaged a number of times directly by the sensor itself. The number of samples used for averaging is called *hardware accelerated average samples*, or *HWAAS* for short. Using this parameter correctly is crucial for reaching the desired signal quality while limiting the memory usage. For static objects, the SNR grows linearly with HWAAS, but keep in mind that so does the measurement time.

See *Radar loop gain (RLG)* for definitions related to SNR, and *Timing* for a detailed description of the timing within a frame.

11 How to configure

11.1 Considerations

Start by asking yourself the following questions related to your application:

- What range do we need to cover, from r_{near} to r_{far}?
- Do we need to distinguish multiple objects? If so, how close could they be?
- Do we need to measure distance to objects? If so, what trueness is needed?
- If objects move, how fast could they go?
- Do we need to measure motions of objects? If so, at what speeds?
- With what rate do we need to obtain a result?
- What are our requirements on overall power consumption?
- What is the farthest distance from the sensor that an object may appear which could result in range ambiguities?

11.2 Range-related parameters

Having these questions and answers in mind, we go though the parameters in order, starting with those related to the range:

- Use the highest **profile** possible for maximum overall power and time efficiency. It is often limited by r_{near} since the *close-in distance (CID)* must be smaller than r_{near}, and higher profiles have a larger CID. If you need to resolve multiple objects, the duration of the pulse must be short enough to give the resolution needed. Finally, lower profiles may give more precise distance measurements.
- The **step length** should be as long as possible to reduce memory usage and decrease measurement time. It is typically limited by two things:
  - Distance measurement trueness: As a rule of thumb, the steps need to be 1/10 to 1/2 of the required trueness. Note that as steps get smaller, other factors such as SNR and pulse duration (profile) have a bigger impact on the general accuracy.
  - The profile: If steps are too long, reflecting objects may fall between points, creating "blind spots" in the range. See Figure 13 for an example.

- From here, the **start point** and **number of points** can be set. Just make sure the points cover r_{near} to r_{far}. Due to the pulse length (profile), the start and end points doesn't necessarily have to pass r_{near} and r_{far}. Again, see Figure 13 for an example of this. However, keep in mind that distance measurements typically cannot be done in the very edge of the range, so you might have to extend it outside r_{near} and r_{far} anyways.
- Set the PRF such that the resulting maximum unambiguous range (MUR) extends beyond the farthest distance an object may appear. Keep it as high as possible since a higher PRF is more power and time efficient overall.

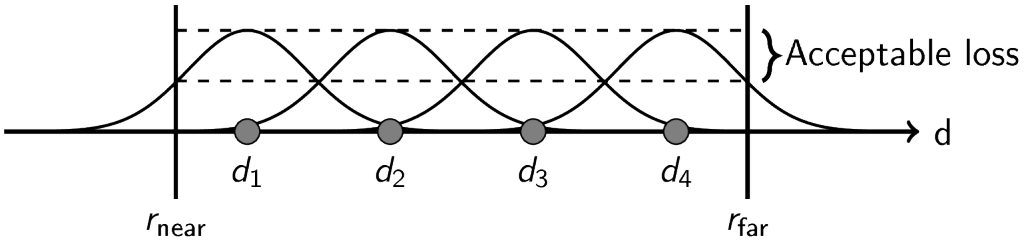


Figure 13: A sketch of setting up the measurement range for efficient coverage of a given area.

11.3 Rate-related parameters

With the range related parameters all set up, we move on to parameters related to sampling rate:

- If you need to estimate velocities, that typically means applying an FFT on a frame over sweeps to produce a distance-velocity (a.k.a. range-Doppler) map. In that case, the **sweeps per frame (SPF)** sets the frequency (velocity) resolution. For e.g. micro- and macro gesture recognition, typical values range from 16 to 64. For more accurate velocity measurements, typical values are much higher ranging from 128 to 2048.

  If you don't need to estimate velocities but still need to detect "fast" motions ($\gtrapprox$ 500Hz), that typically means estimating the energy within a frame. For such cases, e.g. running, walking, waving, typical SPF:s range from 8 to 16.

  If you need to detect "slow" motions or have a mostly static environment, there is no need to use multiple sweeps per frame (SPF), so set it to 1. Such cases include (inter-frame) presence detection and distance measurements.

- For cases where SPF = 1, the **sweep rate** is not applicable. What matters then is setting the **HWAAS** to achieve the needed SNR. Keep in mind that measurement time linearly increases with HWAAS, so keeping it as low as possible is important to manage the overall power consumption.

  For cases where SPF > 1, the **sweep rate** f_s should be adapted to the range of speeds of interest. A good rule of thumb is

  $$f_s \approx \frac{10}{\lambda_{RF}} \cdot |v|_{\max} \approx 2000\text{m}^{-1} \cdot |v|_{\max} \tag{3}$$

  where $|v|_{\max}$ is the possible maximum relative speed between the radar and the object (in m/s).

  In many cases, the most efficient way to achieve the target sweep rate is by adapting the HWAAS. The sweep rate is inversely proportional to the number of HWAAS. It is also possible to directly control the sweep rate, letting the sensor idle between sweeps. However, idling between sweeps is rarely as efficient as idling between frames.

- If power consumption is not an issue, start by using the highest possible **frame rate**. Otherwise, it is crucial to minimize the frame rate to let the sensor idle in a lower idle state as much as possible. See Table 4 for typical values.

Table 4: Typical parameter values for some applications.

Application	Frame rate
Micro gesture recognition	30 - 50 Hz
Medium power presence detection	10 - 80 Hz
Low power presence detection	1 - 5 Hz

11.4 Idle-related parameters and control

Five idling states are available to optimize the power consumption of the sensor. Each state progressively consumes less power, but also increase the wake-up time.

The three most shallow idling states are set through the sensor configuration:

- **READY** - Required state when measuring.
- **SLEEP**
- **DEEP_SLEEP**

The two deepest idling states are set through API calls from the host to the sensor:

- **HIBERNATE**
- **OFF** - Deepest state and longest wake-up.

The first three states are set as a part of the sensor configuration through the parameters **inter sweep idle state** and **inter frame idle state**. Note, the inter frame idle state must be set equal or lower than the inter sweep idle state.

After determining the range and rate related parameters, set the idle states to the deepest possible state, while still being able to maintain the desired sweep and frame rate. If the sweep rate is not set, the sensor will collect sweeps at the highest possible rate. To maximize the sweep rate, set the inter sweep idle state to READY.

The time it takes to transition from SLEEP/DEEP SLEEP to READY impacts the maximum achievable sweep and frame rate. More info regarding idle state transition times can be found *here*.

The wake-up time from HIBERNATE and OFF are integration dependent (SPI communication speed and crystal startup/stablization time) and has to be evaluated for each case. As a rule of thumb, HIBERNATE should be used when the inter frame duration is longer than 15 ms and OFF when the duration is longer than 2 s. These states only affects the inter frame power consumption. The inter sweep idle state needs to be set through the sensor configuration.

Note, HIBERNATE and OFF are only available in the C SDK and not through the Python API.

11.5 Other parameters

- Leave **receiver gain** at the default value and reduce if saturation occurs.
- Leave **enable TX** set (default).

12 Timing

Note

The relationships, equations, and constants described here are approximate, not guaranteed, and may change over RSS releases.

12.1 Notation

τ - A time duration.

f - A frequency or rate. Inverse of T.

T - A period, the time between recurring events. Inverse of f.

N - A number or amount of something.

12.2 Overview

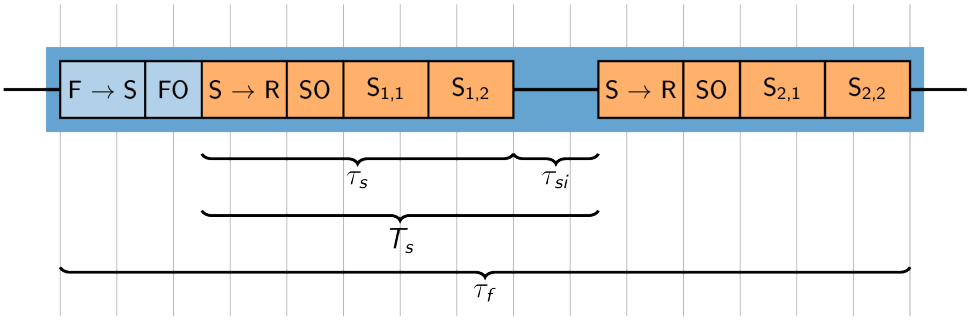


Figure 14: Overview for timing within a frame with two sweeps, in turn consisting of two subsweeps. **Not to scale**, and subsweeps may have different lengths. $F \rightarrow S$ and $S \rightarrow R$ are the times required to transition from the configured *inter frame idle state* to the *inter sweep idle state* state, and *inter sweep idle state* to the *ready* state, respectively. *FO* and *SO* are the fixed overheads on frame and sweep level with duration C_f and C_s respectively. $S_{i,j}$ are subsweeps.

12.3 Sample duration

The time to measure a sample τ_{sample} is the *added* time per HWAAS for every single measured point. In most cases, this accounts for the largest part of the sensor measurement time. The sample duration τ_{sample} depends on the configured *profile* and `PRF` according to Table 5 below.

Table 5: Approximate sample durations τ_{sample} for all profile and PRF f_p combinations.

Profile \ PRF	19.5 MHz	15.6 MHz	13.0 MHz	8.7 MHz	6.5 MHz	5.2 MHz
1	1487 ns	1795 ns	2103 ns	3026 ns	3949 ns	4872 ns
2	N/A	1344 ns	1600 ns	2369 ns	3138 ns	3908 ns
3	N/A	1026 ns	1231 ns	1846 ns	2462 ns	3077 ns
4	N/A	1026 ns	1231 ns	1846 ns	2462 ns	3077 ns
5	N/A	1026 ns	1231 ns	1846 ns	2462 ns	3077 ns

12.4 Point duration

The total time it takes to measure a single distance point in a single (sub)sweep is

$$\tau_{\text{point}} \approx N_a \cdot \tau_{\text{sample}} + \tau_{\text{point overhead}} \tag{4}$$

where N_a is the configured HWAAS (number of averages), and $\tau_{\text{point overhead}}$ is given by Table 6 below.

Table 6: Approximate durations of the point overhead $\tau_{\text{point overhead}}$ for all profile and PRF f_p combinations.

Profile \ PRF	19.5 MHz	15.6 MHz	13.0 MHz	8.7 MHz	6.5 MHz	5.2 MHz
1	1744 ns	2102 ns	2462 ns	3539 ns	4615 ns	5692 ns
2	N/A	1612 ns	1920 ns	2844 ns	3766 ns	4689 ns
3	N/A	1282 ns	1539 ns	2308 ns	3077 ns	3846 ns
4	N/A	1282 ns	1539 ns	2308 ns	3077 ns	3846 ns
5	N/A	1282 ns	1539 ns	2308 ns	3077 ns	3846 ns

12.5 Subsweep duration

The total time it takes to measure a single subsweep, including the time it takes to initialize the measurement is

$$\tau_{\text{subsweep}} \approx N_d \cdot \tau_{\text{point}} + \underbrace{3 \cdot \tau_{\text{sample}} + C_{\text{subsweep}}}_{\text{overhead}} \tag{5}$$

where N_d is the configured number of distances (`num_points`), and C_{subsweep} is a fixed overhead, see section *Fixed overheads and high speed mode (HSM)*.

12.6 Sweep duration

The time to measure a sweep τ_s is the total time it takes to measure a single sweep, including all configured subsweeps and transitioning from the configured inter sweep idle state.

$$\tau_s \approx \tau_{S\to R} + \sum(\tau_{\text{subsweep}}) + C_s \tag{6}$$

where $\tau_{S\to R}$ is the *sweep transition time*, and C_s is a fixed overhead, see section *Fixed overheads and high speed mode (HSM)*.

The sweep transition time $\tau_{S\to R}$ is the time required to transition from the configured *inter sweep idle state* to the *ready* state. See Table 7 below.

Idle state transition times

Table 7: Approximate transition times between the idle states.

From \ To	Deep sleep	Sleep	Ready
Deep sleep	0μs	615μs	670μs
Sleep	N/A	0μs	55μs
Ready	N/A	N/A	0μs

12.7 Sweep period

The sweep period T_s is the time between the start of two consecutive sweeps in a frame.

If the *sweep rate* is not set, the sweep period will be equal to the sweep duration; $T_s = \tau_s$.

If the *sweep rate* is set, the sensor will idle in the configured *inter sweep idle state* between sweeps. This idle time is called the *inter sweep idle time*, τ_{si}.

$$T_s = \frac{1}{f_s} = \tau_s + \tau_{si} \tag{7}$$

12.8 Frame duration

The time to measure a frame τ_f is the total time it takes to measure a frame, including all sweeps and transitioning from the configured inter frame idle state.

$$\tau_f \approx \tau_{F\to S} + (N_s - 1)\cdot T_s + \tau_s + C_f \tag{8}$$

where $\tau_{F\to S}$ is the *frame transition time*, N_s is the SPF (sweeps per frame), and C_f is a fixed overhead, see section *Fixed overheads and high speed mode (HSM)*.

The frame transition time $\tau_{F\to S}$ is the time required to transition from the configured *inter frame idle state* to the *inter sweep idle state* state. See Table 7 above.

12.9 Frame period

The frame period T_f is the time between the start of two consecutive frames. The sensor will idle in the configured *inter frame idle state* between frames. This idle time is called the *inter frame idle time*, τ_{fi}.

$$T_f = \frac{1}{f_f} = \tau_f + \tau_{fi} \tag{9}$$

In most cases, the sensor will not measure (again) until the host commands it to. This means that the frame period, and thus also the *inter frame idle time*, is given by how the host controls the sensor.

If the *frame rate* is not set, the sensor will measure immediately once the host commands it to. Thus, the frame period will be larger than the frame duration; $T_f > \tau_f$.

If the *frame rate* f_f is set, the sensor will continue to idle until Eq. 9 is met.

12.10 Fixed overheads and high speed mode (HSM)

The fixed overheads for subsweep, C_{subsweep}, sweep, C_s, and frame, C_f, are dependent on if high speed mode is activated or not. High speed mode means that the sensor is configured in a way where it can optimize its measurements to obtain as high sweep rate as possible. In order for the sensor to operate in high speed mode, the following configuration constraints apply:

- `continuous_sweep_mode`: False
- `inter_sweep_idle_state`: READY
- `num_subsweeps`: 1
- `profile`: 3-5

Note that the default RSS Service configuration comply with these constraints which means that high speed mode is activated by default.

The fixed overheads can be seen in Table 8 below.

Table 8: Approximate fixed overhead times.

Mode \ Overhead	C_{subsweep}	C_s	C_f
Normal	22μs	10μs	4μs
High speed	0μs	0μs	36μs

13 Detectors

Detectors use the service data to produce a result that can be used by the application. User guides for the different Detectors are available at a121_algorithms and the Detectors are also available in the Exploration Tool.

13.1 Distance detector

This is a distance detector algorithm built on top of the Sparse IQ service, where the filtered sweep is compared to a threshold to identify one or more peaks, corresponding to objects in front of the radar. More details about the detector are found at /exploration_tool/algo/a121/detectors/distance_detection.

13.2 Presence detector

Detects changes in the environment over time based on data from the Sparse IQ service to determine human presence. More details about the detector are found at /exploration_tool/algo/a121/detectors/presence_detection.

14 In-depth topics

This section provides insight into some more in-depth topics.

14.1 Figure of Merits

This page describes and defines Figure of Merits (FoM).

Radar loop gain (RLG)

Signal-to-noise ratio (SNR)

Let $x(f,s,d)$ be a (complex) point at a distance d where the number of distances is N_d, in a sweep s where the number of sweeps per frame is N_s, in a frame f where the number of frames collected is N_f. Further, let x_S be data collected with a known reflector in the range. This is used for the signal part of the SNR. Correspondingly, let x_N be the data collected with the transmitter (TX) disabled. This is used for the noise part of the SNR.

The frame/sweep distinction is not necessary for the SNR definition. The frames can simply be concatenated: $x(f,s,d) \rightarrow y(s',d)$ where $N_{s'} = N_f \cdot N_s$.

Let the signal power $S = \max_d(\text{mean}_{s'}(|y_S|^2))$.

Let the noise power $N = \text{mean}_{s',d}(|y_N|^2)$. This is assuming $E(y_N) = 0$ and $V(y_N)$ is constant over (s',d), where E is the expectation value and V is the variance.

Finally, let $\text{SNR} = S/N$. The SNR in decibel $\text{SNR}_{\text{dB}} = 10 \cdot \log_{10}(S/N)$.

Note

The expectation value of the SNR is **not** affected by the total number of sweeps $N_{s'}$ nor the number of distances N_d measured.

Radar loop gain (RLG)

The radar loop gain is given by: $\text{RLG}_{\text{dB}} = \text{SNR}_{\text{dB}} - \text{RCS}_{\text{dB}} + 40\log_{10}(d)$ where SNR is the signal-to-noise ratio, RLG is the radar loop gain, RCS is the radar cross-section, and d is the distance to the target with the given RCS.

Note

Of the configuration parameters, RLG only depends on *profile* and HWAAS. Excluding the configuration, RLG may vary on other factors such as per unit variations, temperature, and the hardware integration.

Base RLG

Base RLG is the RLG (equivalent) for HWAAS = 1.

SNR and therefore also RLG scales linearly with HWAAS, meaning a 3dB increase for every doubling of HWAAS.

Radar loop gain per time (RLG/t)

Measurement time is not accounted for in the RLG metrics. To account for it, we define the *radar loop gain per time (RLG/t)*:

$$\text{RLG/t}_{\text{dB}} = (\text{RLG}_{\text{dB}} \mid \text{HWAAS} = 1) - 10\log_{10}(\tau_{\text{sample}}) \tag{10}$$

where $(\text{RLG}_{\text{dB}} \mid \text{HWAAS} = 1)$ is the *base RLG*, and τ_{sample} is the *sample duration*.

Note

RLG/t also depends on `PRF`. Higher PRF:s have higher RLG/t, which are generally more efficient.

Radial resolution

Radial resolution is described by the full width at half maximum (FWHM) envelope power.

Let $x(f,s,d)$ be a (complex) point at a radial distance d where the number of distances is N_d, in a sweep s where the number of sweeps per frame is N_s, in a frame f where the number of frames collected is N_f.

Then, let the (average) envelope power

$$y(d) = |\text{mean}_{f,s}(x)|^2 \tag{11}$$

The FWHM of y is what describes the radial resolution.

Distance

Preliminaries

Let $d_{est}(f,d)$ be the estimated distance to a target located at distance d, formed by processing frame f in accordance with the steps outline in the distance detector documentation. f is a single frame in a set of frames of size N_f.

Next, let $e(f,d) = d_{est}(f,d) - d$ be the estimation error of a single frame/measurement.

Lastly, form the mean error by averaging over the frames, $\bar{e}(d) = \text{mean}_f(e(f,d))$.

$\bar{e}(d)$ describes the average error for a single sensor. The metrics calculated in the following sections are based on data from a set of sensors. To indicate what sensor the mean error is associated with, the argument s is added, $\bar{e}(d,s)$.

Accuracy

The distance estimation accuracy is characterized through the following two sets of metrics:

- Mean error(μ) and standard deviation(σ).
- Mean absolute error(MAE).

Mean and standard deviation

The mean error for a set of sensors is given by $\mu = \text{mean}_{d,s}(\bar{e}(d,s))$.

The standard deviation for a set of sensors is given by $\sigma = \text{std}_{d,s}(\bar{e}(d,s))$.

Mean absolute error

The mean absolute error for a set of sensor is given by $\text{MAE} = mean_{d,s}(|\bar{e}(d,s)|)$.

Linearity

Linearity refers to the variation in the distance estimate error as a function of the distance to the target.

The distance linearity is characterized through the mean of the standard deviation of the estimation error over a number of distances, $\sigma = \text{mean}_s(\text{std}_d(\bar{e}(d,s)))$.

The distance linearity is evaluated over two sets of distances:

- Micro: A number of distances within a few wavelengths.
- Macro: A number of distances over many wavelengths.

Temperature sensing

The accuracy of the built-in temperature sensor is described by the *relative deviation*:

$$k = \left| \frac{\hat{x} - x}{x} \right| \tag{12}$$

where x is the actual temperature change and $\hat{x}$ is the measured temperature change.

The evaluated temperature span is typically the range from -40°C to 105°C.

Note

The built-in temperature sensor is not designed for *absolute* measurements and should therefore not be used for that. For this reason, the absolute accuracy is not described as a FoM.

14.2 Continuous sweep mode (CSM)

With CSM, the sensor timing is set up to generate a continuous stream of sweeps, even if more than one sweep per frame is used. The interval between the last sweep in one frame to the first sweep in the next frame becomes equal to the interval between sweeps within a frame (given by the sweep rate).

It ensures that:

$$\text{frame rate} = \frac{\text{sweep rate}}{\text{sweeps per frame}}$$

While the frame rate parameter can be set to approximately satisfy this condition, using CSM is more precise.

If only one sweep per frame is used, CSM has no use since a continuous stream of sweeps is already given (if a fixed frame rate is used).

The main use for CSM is to allow reading out data at a slower rate than the sweep rate, while maintaining that sweep rate continuously.

Note that in most cases, *Double buffering* must be enabled to allow high rates without delays.

Examples of where CSM is used are the Vibration measurement app and the Phase tracking app. In both cases, it is desirable to have a continuous stream of sweeps at a fixed rate with a configurable frame rate.

CSM is enabled through the sensor configuration parameter `continuous_sweep_mode`.

Constraints:

- `frame_rate` must be set to unlimited (None).
- `sweep_rate` must be set (> 0).
- `inter_sweep_idle_state` must be set equal to `inter_frame_idle_state`.

14.3 Loopback

Loopback refers to the process of measuring the generated pulse electronically on the chip by routing it directly to the receiver, rather than transmitting the energy out into the air.

The easiest way to get familiar with the loopback measurement is through the Sparse IQ service in the Exploration Tool. The feature is enabled/disabled using the Sensor Configuration parameter `enable_loopback`.

The following graph shows the measured amplitude and phase with loopback enabled.

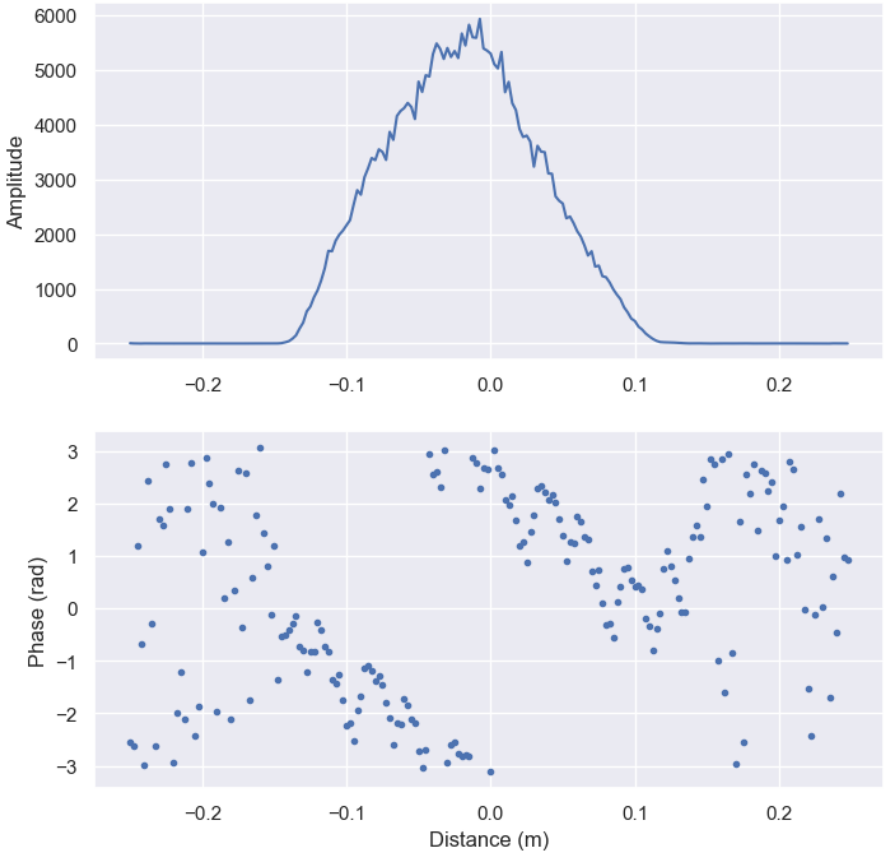


Figure 15: Upper graph: Amplitude of loopback measurement. Lower graph: Phase of loopback measurement.

As can be seen, the result looks like a regular measurement with amplitude and phase. The loopback measurement has the following key features:

#1 - The center of the envelope is located close to zero distance.

#2 - The phase pattern of a loopback measurement and a regular measurement are highly correlated. Phase pattern refers to the series of phase values measured over the defined distances points.

#3 - Timing variations originating from sensor-sensor variations has the same impact on loopback measurements and regular measurements. Timing variations refers to slight shifts of the measured envelope in the distance domain.

#4 - As no energy is transmitted into the air, the measurement is not effected by the surroundings and can be performed at any time.

The following sections illustrates how the loopback measurement is utilized to enhance the performance of the system.

Improved distance estimation

Key features **#3** and **#4** are used in the distance detector to improve the distance accuracy over sensor individuals.

The distance detector estimate the distance to an object as the location of the peak amplitude in the measured envelope. Due to sensor-sensor variations and temperature effects, the timing of the measured envelope from two sensors with identical installation can differ slightly. Hence, variation in envelope timing translates into an error in the estimated distance.

As noted in key feature #3, the envelope timing variation also impacts the loopback measurement. The distance detector takes advantage of this correlation through the implementation of an offset error compensation. The compensation takes the location of the envelope peak amplitude of a loopback measurement as input and outputs an offset value, applied to the estimated distance.

Key feature #4 allows the compensation to be performed at any time, without any considerations to the sensors surroundings.

Phase jitter reduction

Key features **#2** and **#4** can be used to reduce phase jitter of the measured points.

As stated in key feature #2, the phase of a regular measurement and a loopback measurement is highly correlated. This implies that the phase jitter of a regular measurement at any given time can be estimated through a loopback measurement, as the latter is not impacted by the sensor surroundings, according to key feature #4.

The distance detector takes advantage of this concept to achieve a more stable distance estimate when measuring close to the sensor, referred to as a close range measurement. For details, see the Distance detector documentation.

The concept behind the close range measurement strategy is to first characterize the direct leakage and then coherently subtract it from the signal to isolate the signal component of interest. The phase jitter introduce unwanted residuals in the result after subtraction and makes the distance estimate less robust.

The distance detector is configured with a first subsweep containing a regular measurement, used for the distance estimation. It is followed by a second subsweep containing a single point with loopback enabled, used in the process of reducing the impact of the phase jitter.

The direct leakage is characterized by storing a snapshot of the complex values in the first subsweep. At the time of characterization, it is paired with the loopback measurement in the second subsweep, referred to as the loopback phase reference.

As the loopback measurement is not impacted by the sensor surroundings, any deviation in phase from the loopback phase reference is due to phase jitter. The difference is referred to as the instantaneous phase jitter.

Before performing the coherent subtraction, the argument of the complex samples of the characterized direct leakage are adjusted by the amount reflected by the instantaneous phase jitter, to mitigate the impact of the phase jitter.

The full procedure results in a vector with reduced residuals originating from the phase jitter and yields a more robust distance estimate.

Phase enhancement

Key feature **#2** and **#4** are used to achieve phase coherent data in the distance domain.

Phase coherency in the distance domain enables coherent distance filtering, allowing for increased SNR through data processing. For details regarding distance filtering, see the see the Distance detector documentation.

The first step of the phase enhancement process takes place during the sensor calibration where a loopback measurement is performed to quantify the phase pattern over a fixed distance interval. Next, the result from the calibration is applied to subsequent measurements, where the argument of the complex samples are adjusted according to the quantified phase pattern.

The following graph shows the phase and envelope of a measurement against a single target, with and without phase enhancement enabled.

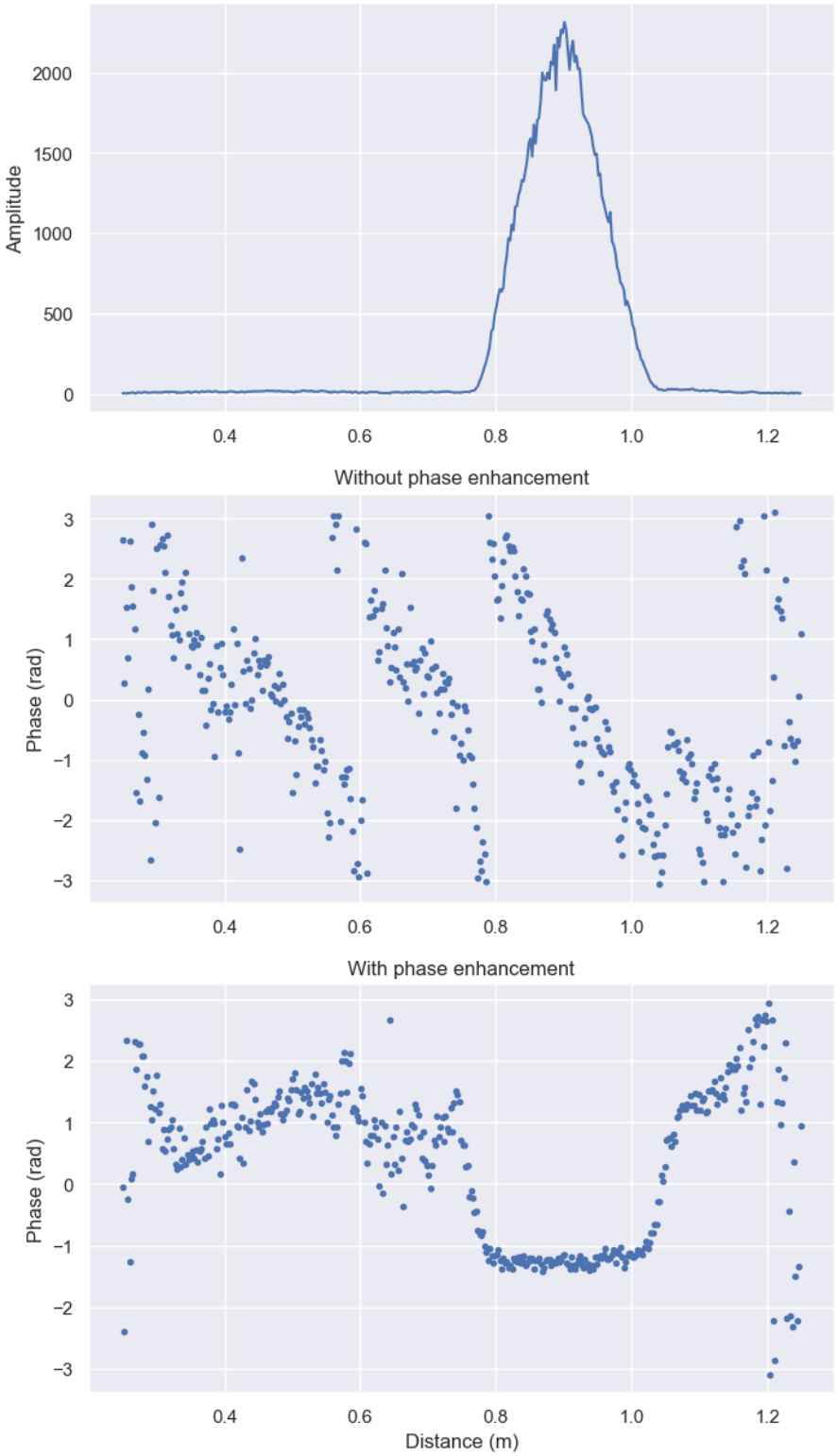


Figure 16: Upper graph: Amplitude of measured sweep. Middle graph: Phase of measured sweep with phase enhancement disabled. Lower graph: Phase of measured sweep with phase enhancement enabled.

The phase enhancement feature is implemented as a part of RSS and can be enabled/disabled through the API. The easiest way to get familiar with the feature is through Exploration Tool, where it can be enabled/disabled.

14.4 Control

This page describes how the A121 is controlled from its host. The full functionality is offered in the C SDK, while Exploration Tool only provides a subset of features.

Fundamentals

In the basic way of operating the A121, the process goes through 3 repeating stages:

1. **Measure**: The host signals the sensor to start measuring. The interrupt pin will go low, and then high again when the sensor completes the measurement.
2. **Wait for interrupt**: The host waits for the interrupt indicating that said measurement is finished.
3. **Read**: The host reads out the data from said measurement.

Figure 17 below shows this operation.

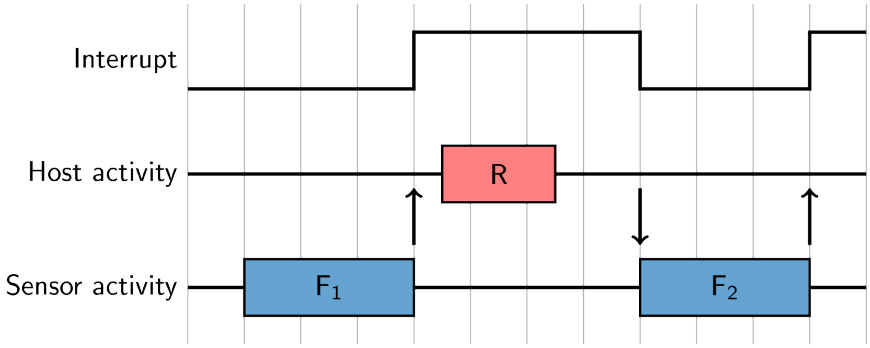


Figure 17: Illustration of basic control flow of the A121. The top part shows the interrupt pin. The middle part shows what the host is doing, where a red box denoted with an R is a *read*. The bottom part shows what the sensor is doing, where a blue box denoted with a F is a frame measurement. A down arrow shows a *measure* call, and an up arrow shows a completed measurement.

Note that in the case shown in Figure 17, the rate of frame measurements is effectively given by the rate of the *measure* call. The A121 is also capable of triggering itself, giving an extremely accurate rate. This is done by setting the `frame_rate` in configuration. In this case, the sensor will not start its measurement until the corresponding time has passed. Figure 18 shows this method of operation.

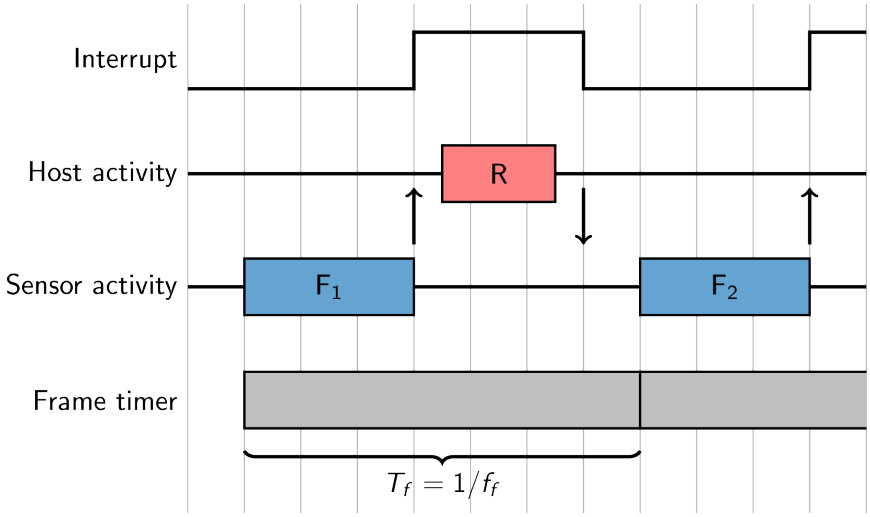


Figure 18: Illustration of a control flow of the A121 where the *frame rate* is set. For context, see Figure 17. The bottom part shows the frame timer set up according to the frame rate.

In the case shown in Figure 18, we can see that the *measure* call, making the interrupt go low, happens before the timer

ends (which triggers the frame measurement start). If the host responds with the call **after** the timer ends, the frame measurement will be delayed, as illustrated in Figure 19 below.

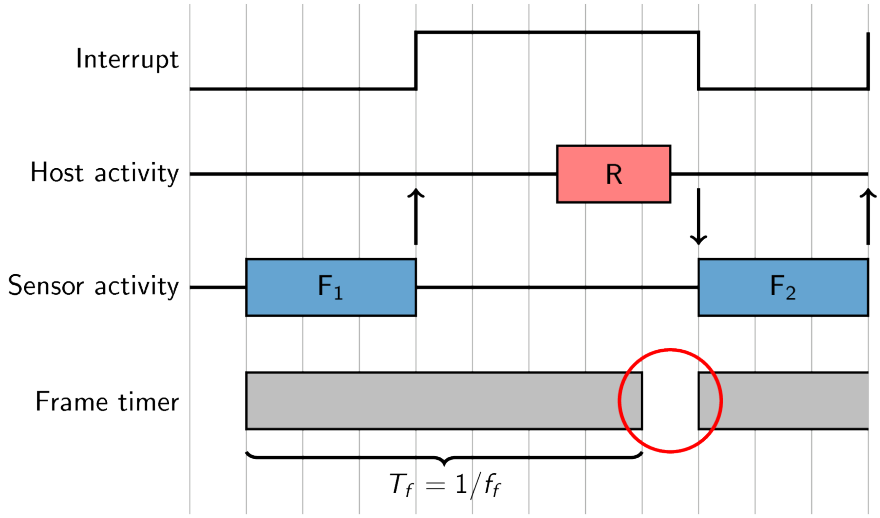


Figure 19: Illustration of a control flow of the A121 where the *frame rate* is set and the host responds late. This makes the frame *delayed*. For context, see Figure 18.

Double buffering

The A121 is capable of operating in a *double buffering* mode where two data buffers (A & B) are used to be able to read out data and measure at the same time. This feature is enabled by the configuration parameter `double_buffering`. Commonly used together with *Continuous sweep mode (CSM)*. Using this mode changes the control flow slightly, in that the sensor will be one frame measurement ahead of the host at all times. Other than that, the principles for rate control are the same.

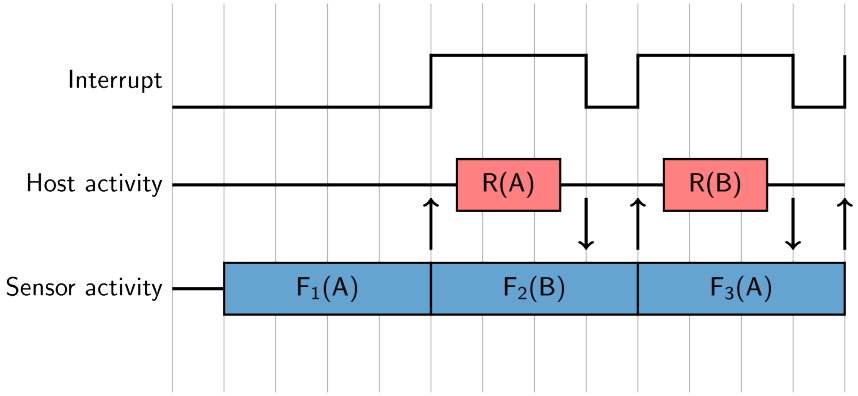


Figure 20: Illustration of a control flow of the A121 using *double buffering*.

Double buffering is typically used for one of two reasons:

1. Enabling *Continuous sweep mode (CSM)*, where the sensor timing is set up to generate a continuous stream of sweeps.
2. Giving the host more time to read out the data before a subsequent frame measurement.

Part III

A111

15 Profiles

The first step is to select pulse length profile to optimize on either depth resolution or radar loop gain, or in terms of use cases, optimized for multiple objects/close range or for weak reflections/long range, respectively.

Depth resolution, d_{res}, is the ability to resolve reflections which are closely spaced, and hence depends on t_{pulse} according to

$$d_{res} \approx \frac{t_{pulse} v}{2} \tag{13}$$

Figure 21 illustrates how the ability to resolve closely spaced reflections can be improved by decreasing t_{pulse}. On the other hand, decreasing t_{pulse} means that the total energy in the pulse is decreased and hence decrease the SNR in the receiver, this is the trade-off that is made by selecting between the five profiles. Each service can be configured with five different pulse length profiles (see Table 9), where

- shorter pulses provides higher distance resolution at the cost of a reduced SNR
- longer pulses provides higher SNR at a cost of reduced depth resolution

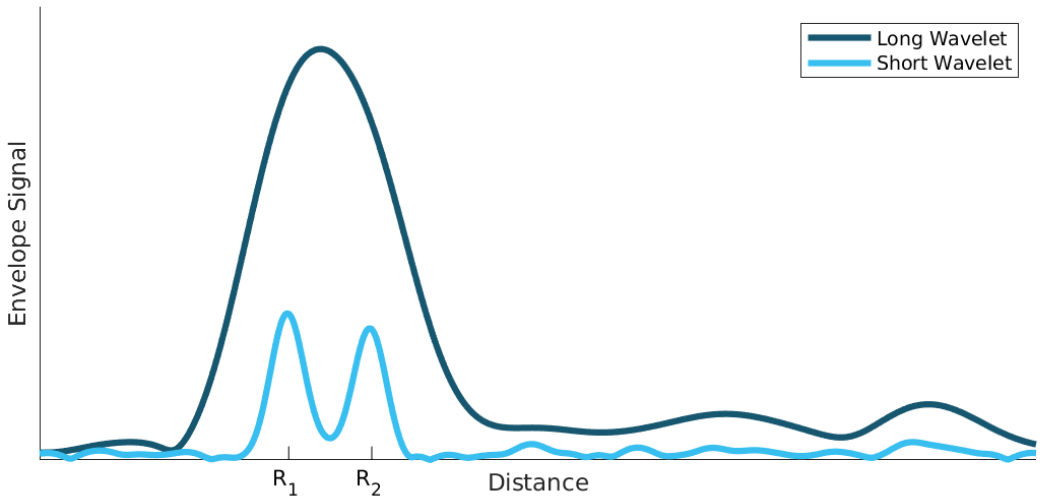


Figure 21: Illustration of received signal containing 2 echoes. A longer pulse increases the radar loop gain, but also limits the depth resolution. The displayed data corresponds to the two setups in Figure 22.

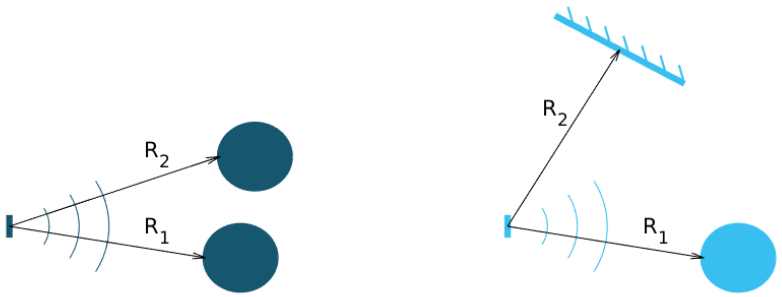


Figure 22: Illustration of scenarios that can produce the data in Figure 21. A strong reflector, such as a flat metallic surface, can give a moderate radar signal if the angle to the radar is high. R_1 is identical in the two illustrations as well as R_2.

Optimizing on depth resolution also means that close-in range performance is improved. The A111 sensor has both the Tx and Rx antenna integrated and since they are so closely spaced, there will be leakage between the two antennas. This means that any object close to the sensor will have to be filtered from this static leakage. The ability to do this is improved if a short t_{pulse} is used, as illustrated in Figure 23.

If angular information is needed one possibility is to mechanically move the sensor to scan an area and produce a synthetic aperture radar (SAR). One such case is for autonomous robots using sensor input for navigation. Another option is to use multiple A111 sensors and merge data from them to calculate the position of the object by trilateration. This can be achieved by running the sensors sequentially and merge the data in the application.

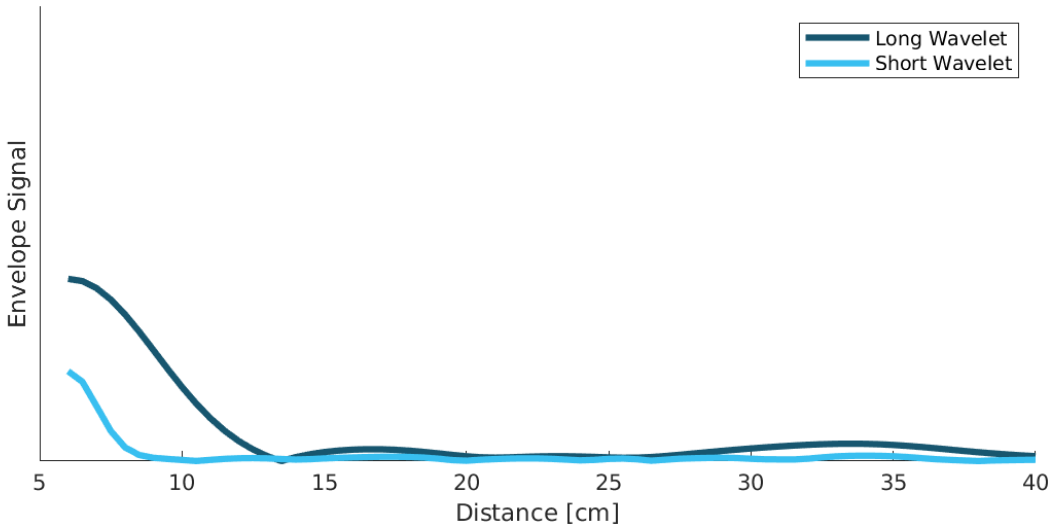


Figure 23: Illustration of how the leakage between the Tx and Rx antenna will appear in the Envelope Service data for Profile 1 and Profile 2 pulse lengths.

Table 9: **Rough** comparison of the envelope service behavior for different profiles.

Profile	Relative SNR improvement [dB]	Direct leakage [m]
Profile 1	0	~0.06
Profile 2	~7	~0.10
Profile 3	~11	~0.18
Profile 4	~13	~0.36
Profile 5	~16	~0.60

16 Typical ranges for different objects

In Table 10 and Table 11 the visibility for a range of objects with common shapes (cylinder, plate, etc.) and of varying reflectivity, i.e. materials, is shown. Objects are at normal incidence and the governing system parameters are σ, γ, and C, as shown in Eq. 16. The envelope service was used to collect the data with Profile 2. The object counts as distinguishable from the noise with a SNR > 10 dB (Y), barely visible between 5 dB and 10 dB (-) and not visible with a SNR < 5 dB (N). The range can be further increased based on the configuration of the sensor, as described in *Configuration overview* and by optimizing the physical integration, as will be described in *Physical integration*. As an example for such an optimization Table 10 shows results with an added radar Fresnel lens.

Table 10: Typical ranges using the envelope service and Profile 2, **without radar lens**.

Object	0.5 m	1 m	2 m	5 m	7 m
Corner reflector (a = 4 cm)	Y	Y	Y	Y	N
Planar water surface	Y	Y	Y	Y	Y
Disc (r = 4 cm)	Y	Y	Y	Y	Y
Cu Plate (10x10 cm)	Y	Y	Y	Y	Y
PET plastic Plate (10x10 cm)	Y	Y	Y	Y	–
Wood Plate (10x10 cm)	Y	Y	–	N	N
Cardboard Plate (10x10 cm)	Y	Y	Y	N	N
Al Cylinder (h = 30, r = 2 cm)	Y	Y	–	N	N
Cu Cylinder (h = 12, r = 1.6 cm)	Y	Y	Y	N	N
PP plastic Cylinder (h = 12, r = 1.6 cm)	Y	N	N	N	N
Leg	Y	Y	–	N	N
Hand (front)	Y	Y	N	N	N
Torso (front)	Y	Y	Y	N	N
Head	Y	Y	N	N	N
Glass with water (h = 8.5, r = 2.7 cm)	Y	Y	N	N	N
PET Bottle with water (h = 14, r = 4.2 cm)	Y	Y	N	N	N
Football	Y	Y	N	N	N

Table 11: Typical ranges using the envelope service and Profile 2, **with 7 dB radar lens**.

Object	0.5 m	1 m	2 m	5 m	7 m
Corner reflector (a = 4 cm)	Y	Y	Y	Y	Y
Planar water surface	Y	Y	Y	Y	Y
Disc (r = 4 cm)	Y	Y	Y	Y	Y
Cu Plate (10x10 cm)	Y	Y	Y	Y	Y
PET plastic Plate (10x10 cm)	Y	Y	Y	Y	Y
Wood Plate (10x10 cm)	Y	Y	Y	Y	N
Cardboard Plate (10x10 cm)	Y	Y	Y	Y	–
Al Cylinder (h = 30, r = 2 cm)	Y	Y	Y	Y	–
Cu Cylinder (h = 12, r = 1.6 cm)	Y	Y	Y	Y	–
PP plastic Cylinder (h = 12, r = 1.6 cm)	Y	Y	Y	N	N
Leg	Y	Y	Y	Y	N
Hand (front)	Y	Y	Y	N	N
Torso (front)	Y	Y	Y	Y	N
Head	Y	Y	Y	–	N
Glass with water (h = 8.5, r = 2.7 cm)	Y	Y	Y	–	N
PET Bottle with water (h = 14, r = 4.2 cm)	Y	Y	Y	N	N
Football	Y	Y	Y	N	N

17 Configuration overview

The Acconeer A111 is highly configurable and can operate in many different modes where parameters are tuned to optimize the sensor performance for specific use cases.

17.1 Signal averaging and gain

In addition to the Profile configuration parameter, two main configuration parameters are available in all Services to optimize the signal quality:

- Hardware Accelerated Average Samples (HWAAS) is related to the number of pulses averaged in the radar to produce one data point. A high number will increase the radar loop gain but each sweep will take longer to acquire and therefore limit the maximum update rate.
- The gain of the amplifiers in the sensor. Adjusting this parameter so the ADC isn't saturated and at the same time the signal is above the quantization noise is necessary. A gain figure of 0.5 is often a good start.

17.2 Sweep and update rate

A sweep is defined as a distance measurement range, starting at the distance *start range* and continues for *sweep length*. Hence, every sweep consists of one or several distance sampling points.

A number of sweeps N_s are sampled after each other and the time between each sweep is T_s, which is configurable. We usually refer to this as the *update rate* $f_s = 1/T_s$.

In addition, the sparse service introduces a concept of frames defined here.

17.3 Repetition modes

RSS supports two different *repetition modes*. They determine how and when data acquisition occurs. They are:

- **On demand**: The sensor produces data when requested by the application. Hence, the application is responsible for timing the data acquisition. This is the default mode, and may be used with all power save modes.
- **Streaming**: The sensor produces data at a fixed rate, given by a configurable accurate hardware timer. This mode is recommended if exact timing between updates is required.

Note, Exploration Tool is capable of setting the update rate also in *on demand* mode. Thus, the difference between the modes becomes subtle. This is why *on demand* and *streaming* are called *host driven* and *sensor driven* respectively in Exploration Tool.

17.4 Power save modes

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes, see Table 12. The different states differentiate in current dissipation and response latency, where the most current consuming mode *Active* gives fastest response and the least current consuming mode *Off* gives the slowest response. The absolute response time and also maximum update rate is determined by several factors besides the power save mode configuration. These are profile, length, and hardware accelerated average samples. In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. Nonetheless, the relation between the power save modes are always kept such that *Active* is fastest and *Off* is slowest.

Another important aspect of the power save mode is when using the service in repetition mode Streaming. In streaming mode the service is also configured with an update rate at which the sensor produces new data. The update rate is maintained by the sensor itself using either internally generated clock or using the externally applied clock on XIN/XOUT pins. Besides the fact that power save mode *Active* gives the highest possible update rate, it also gives the best update rate accuracy. Likewise, the power save mode *Sleep* gives a lower possible update rate than *Active* and also a lower update rate accuracy. Bare in mind that also in streaming mode the maximum update rate is not only determined by the power save mode but also profile, length, and hardware accelerated average samples. Power save mode *Off* and *Hibernate* is not supported in streaming mode since the sensor is turned off between its measurements and thus cannot keep an update rate. In addition, the power save mode *Hibernate* is only supported when using Sparse service.

Table 12 concludes the power save mode configurations.

Table 12: Power save modes.

Power save mode	Current consumption	Response time	Update rate accuracy
Off	Lowest	Longest	Not applicable
Hibernate	...	...	Not applicable
Sleep	...	...	Worst
Ready	...	...	...
Active	Highest	Shortest	Best

As part of the deactivation process of the service the sensor is disabled, which is the same state as power save mode *Off*.

17.5 Configuration summary

Table 13 shows a list of important parameters that are available through our API and that can be used to optimize the performance for a specific use case, refer to product documentation and user guides for a complete list of all parameters and how to use them.

Table 13: List of sensor parameters

Parameter	Comment
Profile	Selects between the pulse length profiles. Trade off between SNR and depth resolution.
Start	Start of sweep [m].
Length	Length of sweep, independently of Start range [m].
HWAAS	Amount of radar pulse averaging in the sensor.
Receiver gain	Adjust to accommodate received signal level.
Repetition mode	On demand or Streaming.
Update rate	Desired rate at which sweeps are generated [Hz] (in repetition mode Streaming).
Power save mode	Tradeoff between power consumption and rate and accuracy at which sweeps are generated.

18 Services and detectors overview

The RSS provides output at two different levels, Service and Detector. The Service output is pre-processed sensor data as a function of distance. Detectors are built with this Service data as the input and the output is a result, in the form of e.g. distance, presence, angle etc. Services and Detectors currently available are listed in Figure 24.

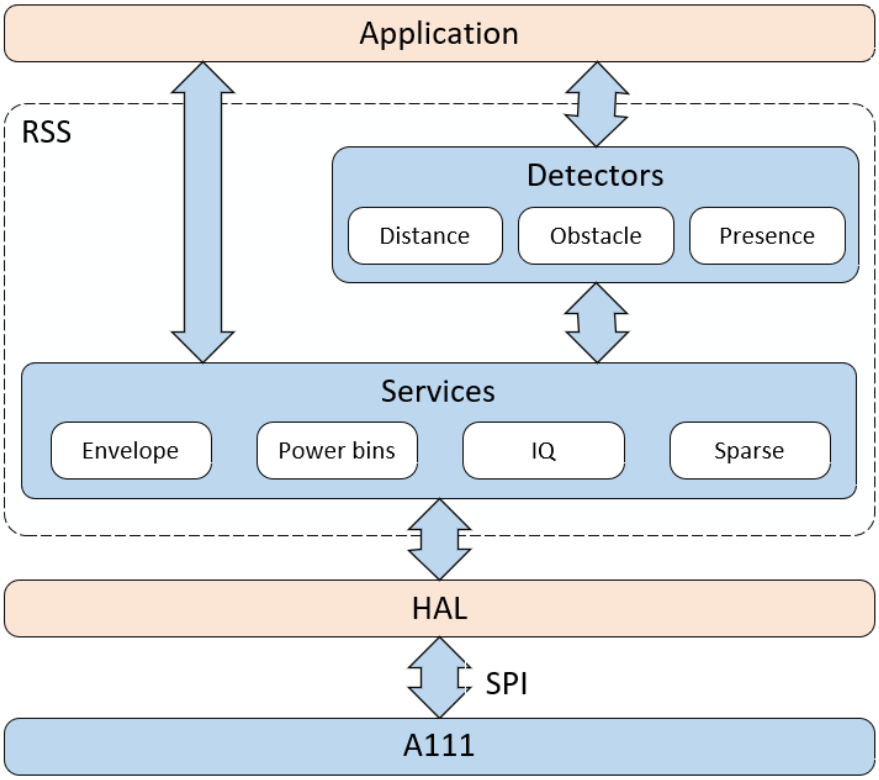


Figure 24: Available Detectors and Services.

Each Detector is built on top of a Service, i.e. you have the possibility to use our out-of-the-box Detectors or develop your own. To select the Service or Detector applicable for your use case it is recommended to use the Exploration Tool to observe the different outputs and understand what they represent. Each Service and Detector also comes with its own user guide, which can be found at acconeer.com.

At developer.acconeer.com, we have several movies showing demos where the Acconeer sensor is used in different use cases. Together with the demo movies, corresponding reference applications are available in our different SDKs at Acconeer developer site. These reference applications are written in C code and use our Services and Detectors, check them out to get inspiration on how to build your product with the Acconeer sensor.

19 Services

The Acconeer A111 sensor can be used in different modes depending on use case. There are currently four such services optimized for different purposes.

Service	Typical use cases
Power Bins	Low complexity implementations, parking sensor
Envelope	Distance measurements, static target scenes
IQ	Obstacle detection, vital sign monitoring
Sparse	Presence detection, hand gesture recognition

19.1 Power Bins

The Power Bins service provides data related to the received energy at different distances from the radar sensor. One Power Bins measurement is performed by transmitting a sequence radar pulses and measuring the energy content in the returning echoes for a set of time delays from pulse transmission.

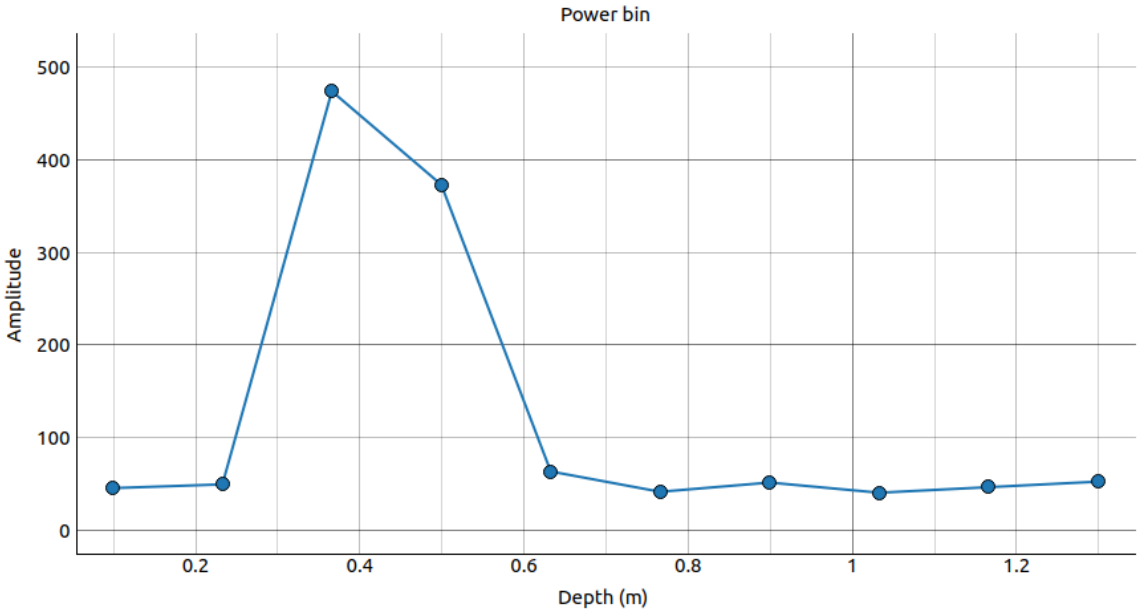
The Power Bins Service is mainly intended for use in low power applications where large objects are measured at short distances from the radar sensor, e.g. in parking sensors mounted on the ground detecting cars. Hence, for use cases with inherently good signal-to-noise ratio the Power Bins service can be used as a low complexity replacement of the *Envelope* service.

The power bins values are calculated using an algorithm optimized for low computational complexity, however this algorithm will not suppress noise as much as the more advanced signal processing algorithms used in the Envelope and IQ Data Services. For use cases with inherently good signal-to-noise ratio the Power Bin service can be used as a low complexity replacement of the *Envelope* service. The Power Bin service outputs coarse estimates of the envelope yielding rough estimates of the range-dependent reflectivity in the target scene. By omitting several of the signal processing steps in the *Envelope* service the computational footprint is significantly reduced, making this service suitable for small platforms.

For use cases with weaker radar echoes, i.e., lower SNR, or when the resolution of the signal is important, we recommend using the Envelope, IQ or Sparse data service instead.

The Power Bins service can be configured with different pulse length profiles, see *Profiles*.

`examples/a111/services/power_bins.py` contains example code on how the Power Bins service can be used.



For further reading on the power bins service we refer to the Power Bins documentation on the Acconeer developer page.

Configuration parameters

`class acconeer.exptool.a111.PowerBinServiceConfig`

`class MUR(`*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

`class PowerSaveMode(`*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

`asynchronous_measurement`

Enabling asynchronous measurements will result in a faster update rate but introduces a risk of interference between sensors.

Type: bool
Default value: True

`bin_count`

The number of bins to be used for creating the amplitude over distance histogram.

Type: int
Default value: 5

`downsampling_factor`

The range downsampling by an integer factor. A factor of 1 means no downsampling, thus sampling with the smallest possible depth interval. A factor 2 samples every other point, and so on. In Envelope and IQ, the finest interval is ~0.5 mm. In Power Bins, it is the same but then further downsampled in post-processing. In sparse, it is ~6 cm.

The downsampling is performed by skipping measurements in the sensor, and therefore gives lower memory usage, lower power consumption, and lower duty cycle.

In sparse, setting a too large factor might result in gaps in the data where moving objects "disappear" between sampling points.

In Envelope, IQ, and Power Bins, the factor must be 1, 2, or 4. In sparse, it must be at least 1. Setting a factor greater than 1 might affect the range end point and for IQ and Envelope, also the first point.

Type: int
Default value: 1

`gain`

The receiver gain used in the sensor. If the gain is too low, objects may not be visible, or it may result in poor signal quality due to quantization errors. If the gain is too high, strong reflections may result in saturated data. We recommend not setting the gain higher than necessary due to signal quality reasons.

Must be between 0 and 1 inclusive, where 1 is the highest possible gain.

Note

When Sensor normalization is active, the change in the data due to changing gain is removed after normalization. Therefore, the data might seen unaffected by changes in the gain, except very high (receiver saturation) or very low (quantization error) gain.

Sensor normalization is not available for the Sparse service, but is enabled by default for the other services - Envelope, IQ, and Power Bins.

Type: float
Default value: 0.5

`hw_accelerated_average_samples`

Number of samples taken to obtain a single point in the data. These are averaged directly in the sensor hardware - no extra computations are done in the MCU.

The time needed to measure a sweep is roughly proportional to the HWAAS. Hence, if there's a need to obtain a higher sweep rate, HWAAS could be decreased. Note that HWAAS does not affect the amount of data transmitted from the sensor over SPI.

Must be at least 1 and not greater than 63.

Type: int
Default value: 10

`maximize_signal_attenuation`

When measuring in the direct leakage (around 0m), this setting can be enabled to minimize saturation in the receiver. We do not recommend using this setting under normal operation.

Type: bool
Default value: False

`mur`

Sets the *maximum unambiguous range* (MUR), which in turn sets the *maximum measurable distance* (MMD).

The MMD is the maximum value for the range end, i.e., the range start + length. The MMD is smaller than the MUR due to hardware limitations.

The MUR is the maximum distance at which an object can be located to guarantee that its reflection corresponds to the most recent transmitted pulse. Objects farther away than the MUR may fold into the measured range. For example, with a MUR of 10 m, an object at 12 m could become visible at 2 m.

A higher setting gives a larger MUR/MMD, but comes at a cost of increasing the measurement time for a sweep. The measurement time is approximately proportional to the MUR.

This setting changes the *pulse repetition frequency* (PRF) of the radar system. The relation between PRF and MUR is

$$\mathrm{MUR} = c/(2 * \mathrm{PRF})$$

where c is the speed of light.

Setting	MUR	MMD	PRF
6	11.5 m	7.0 m	13.0 MHz
9	17.3 m	12.7 m	8.7 MHz

This is an experimental feature.

Default value: MUR.MUR_6

`noise_level_normalization`

With the SW version 2 release, a sensor signal normalization functionality is activated by default for the Power Bins, Envelope, and IQ Service. This results in a more constant signal for different temperatures and sensors. The radar sweeps are normalized to have similar amplitude independent of sensor gain and hardware averaging, resulting in only minor visible effect in the sweeps when adjusting these parameters.

We recommend this setting especially for applications, where absolute radar amplitudes are important, such as when comparing to a previously recorded signal or to a fixed threshold.

More technically, the functionality is implemented to collect data when starting the service, but not transmitting pulses. This data is then used to determine the current sensitivity of receiving part of the radar by estimating the power level of the noise, which then is used to normalize the collected sweeps. In the most low-power systems, where a service is created to collect just a single short sweep before turning off, the sensor normalization can add a non-negligible part to the power consumption.

Please note, that due to the nature of Sparse data, the Sparse service does not support noise level normalization. Instead, normalization during processing is recommended, such as done in the Presence detector.

Type: bool
Default value: True

`power_save_mode`

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes. The modes differentiate in current dissipation and response latency, where the most current consuming mode *Active* gives fastest response and the least current consuming mode *Off* gives the slowest response. The absolute response time and also maximum update rate is determined by several factors besides the power save mode configuration.

In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. The power consumption of the system depends on the actual configuration of the application and it is recommended to test both maximum update rate and power consumption when the configuration is decided.

Power save mode	Current consumption	Response time
Off	Lowest	Longest
Hibernate	...	...
Sleep	...	...
Ready	...	...
Active	Highest	Shortest

Note

Hibernation has limited hardware support. It is not supported by the Raspberry Pi EVK:s and XM112.

Default value: PowerSaveMode.ACTIVE

`profile`

The main configuration of all the services are the profiles, numbered 1 to 5. The difference between the profiles is the length of the radar pulse and the way the incoming pulse is sampled. Profiles with low numbers use short pulses while the higher profiles use longer pulses.

Profile 1 is recommended for:

- measuring strong reflectors, to avoid saturation of the received signal
- close range operation (<20 cm), due to the reduced direct leakage

Profile 2 and 3 are recommended for:

- operation at intermediate distances, (20 cm to 1 m)
- where a balance between SNR and depth resolution is acceptable

Profile 4 and 5 are recommended for:

- for Sparse service only
- operation at large distances (>1 m)
- motion or presence detection, where an optimal SNR ratio is preferred over a high resolution distance measurement

The previous profile Maximize Depth Resolution and Maximize SNR are now profile 1 and 2. The previous Direct Leakage Profile is obtained by the use of the Maximize Signal Attenuation parameter.

Default value: Profile.PROFILE_2

`range_interval`

The measured depth range. The start and end values will be rounded to the closest measurement point available.

The the sweep range is limited to 7.0 m for the default “maximum unambiguous range” setting. To change this limitation, increase “maximum unambiguous range”.

Type: float
Unit: m
Default value: [0.18, 0.78]

`repetition_mode`

The RSS supports two different repetition modes. They determine how and when data acquisition occurs. They are:

- **On demand / host driven**: The sensor produces data when requested by the application. Hence, the application is responsible for timing the data acquisition. This is the default mode, and may be used with all power save modes.
- **Streaming / sensor driven**: The sensor produces data at a fixed rate, given by a configurable accurate hardware timer. This mode is recommended if exact timing between updates is required.

The Exploration Tool is capable of setting the update rate also in *on demand (host driven)* mode. Thus, the difference between the modes becomes subtle. This is why *on demand* and *streaming* are called *host driven* and *sensor driven* respectively in Exploration Tool.

Default value: RepetitionMode.HOST_DRIVEN

`sensor`

The sensor(s) to be configured.

Default value: [1]

`tx_disable`

Disable the radio transmitter. If used to measure noise, we recommended also switching off noise level normalization (if applicable).

Type: bool
Default value: False

`update_rate`

The rate f_f at which the sensor sends frames to the host MCU.

> **Attention**
>
> Setting the update rate too high might result in missed data frames.

In sparse, the maximum possible update rate depends on the *sweeps per frame* N_s and *sweep rate* f_s:

$$\frac{1}{f_f} > N_s \cdot \frac{1}{f_s} + \text{overhead*}$$

* *The overhead largely depends on data frame size and data transfer speeds.*

Type: float
Unit: Hz
Default value: None

19.2 Envelope

The Envelope service provides data related to the received energy at different distances from the radar sensor. One envelope measurement is performed by transmitting a sequence radar pulses and measuring the energy content in the returning echoes for a set of time delays from pulse transmission.

The envelope data is a set of N_D real valued samples represented as $x[d]$, where d is the delay sample index and each value sample has an amplitude $x[d] = A_d$ representing the received energy from a specific distance, d.

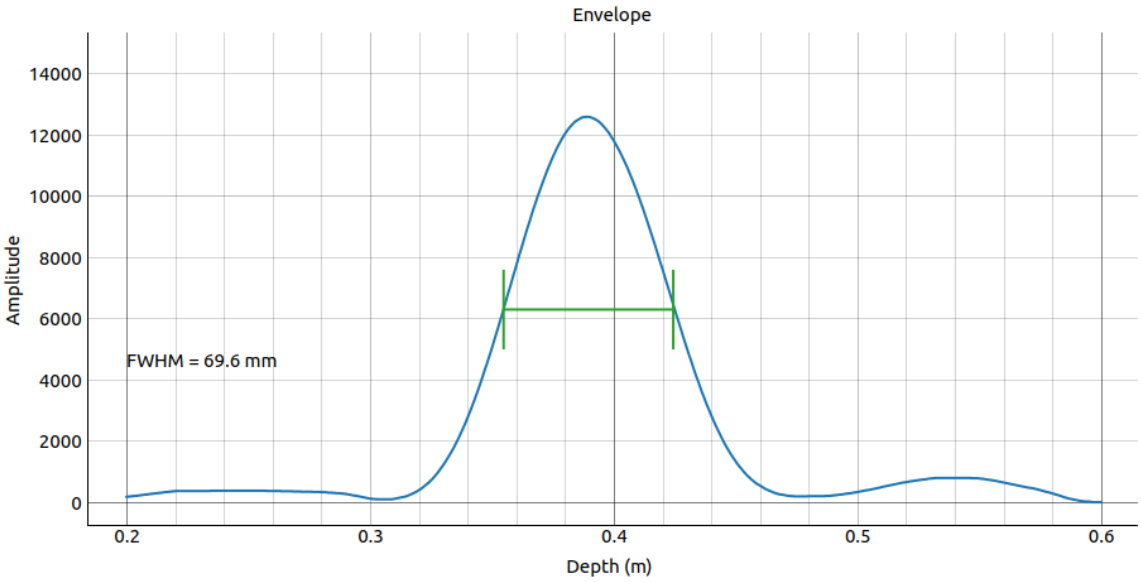
To stabilize the signal and increase the SNR, the sweeps in the Envelope Service can be time filtered. This can be configured by setting the running average factor between 0 and 1, where high number indicates more filtering. The

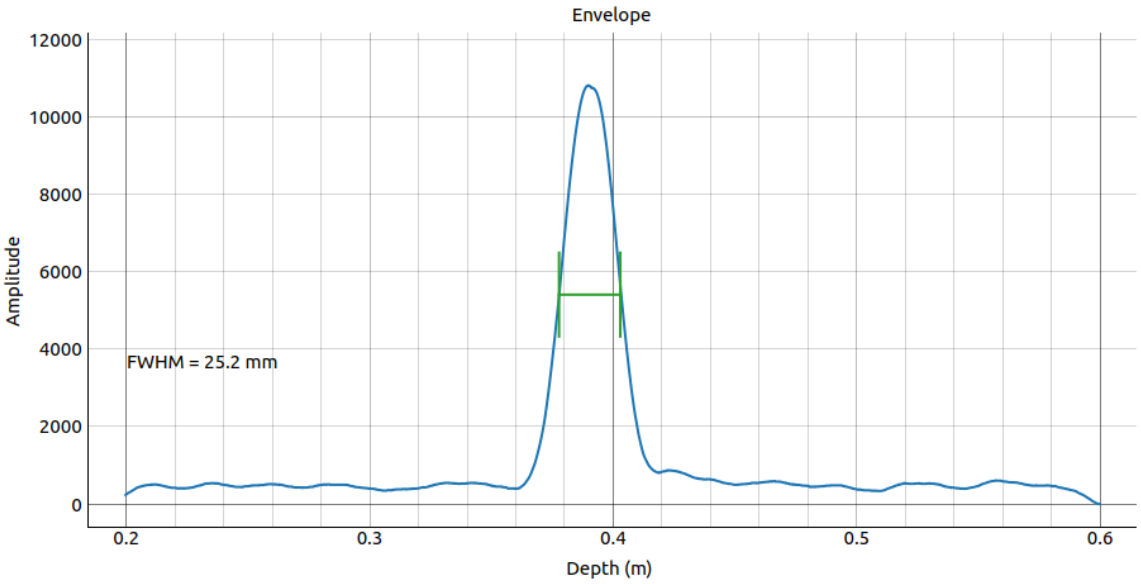
filtering is a standard exponential smoothening filter with default setting of 0.7. Note, for very low update rates below a few hertz, the signal from an object moving fast, or suddenly disappearing, remain in the sweep.

The filtering of data in the Envelope Service applies a low-pass filter in the range dimension. This leads to some filter edge effects in the first few centimeters of the sweep. For very short sweeps, approx. 3 cm for Profile 1 and approx 6 cm for Profile 2-5, these edge effects affects the magnitude of the whole sweep. The *Power Bins*, with only a few bins, is recommended for short sweeps.

The Envelope service can be configured with different pulse length profiles, see *Profiles*.

`examples/a111/services/envelope.py` contains example code on how this service can be used.





For further reading on the envelope service we refer to the Envelope documentation on the Acconeer developer page.

Configuration parameters

`class acconeer.exptool.a111.EnvelopeServiceConfig`

`class MUR`(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

`class PowerSaveMode(`*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

`asynchronous_measurement`

Enabling asynchronous measurements will result in a faster update rate but introduces a risk of interference between sensors.

Type: bool

Default value: True

`downsampling_factor`

The range downsampling by an integer factor. A factor of 1 means no downsampling, thus sampling with the smallest possible depth interval. A factor 2 samples every other point, and so on. In Envelope and IQ, the finest interval is ~0.5 mm. In Power Bins, it is the same but then further downsampled in post-processing. In sparse, it is ~6 cm.

The downsampling is performed by skipping measurements in the sensor, and therefore gives lower memory usage, lower power consumption, and lower duty cycle.

In sparse, setting a too large factor might result in gaps in the data where moving objects "disappear" between sampling points.

In Envelope, IQ, and Power Bins, the factor must be 1, 2, or 4. In sparse, it must be at least 1. Setting a factor greater than 1 might affect the range end point and for IQ and Envelope, also the first point.

Type: int

Default value: 1

`gain`

The receiver gain used in the sensor. If the gain is too low, objects may not be visible, or it may result in poor signal quality due to quantization errors. If the gain is too high, strong reflections may result in saturated data. We recommend not setting the gain higher than necessary due to signal quality reasons.

Must be between 0 and 1 inclusive, where 1 is the highest possible gain.

Note

When Sensor normalization is active, the change in the data due to changing gain is removed after normalization. Therefore, the data might seen unaffected by changes in the gain, except very high (receiver saturation) or very low (quantization error) gain.

Sensor normalization is not available for the Sparse service, but is enabled by default for the other services - Envelope, IQ, and Power Bins.

Type: float

Default value: 0.5

`hw_accelerated_average_samples`

Number of samples taken to obtain a single point in the data. These are averaged directly in the sensor hardware - no extra computations are done in the MCU.

The time needed to measure a sweep is roughly proportional to the HWAAS. Hence, if there's a need to obtain a higher sweep rate, HWAAS could be decreased. Note that HWAAS does not affect the amount of data transmitted from the sensor over SPI.

Must be at least 1 and not greater than 63.

Type: int
Default value: 10

`maximize_signal_attenuation`

When measuring in the direct leakage (around 0m), this setting can be enabled to minimize saturation in the receiver. We do not recommend using this setting under normal operation.

Type: bool
Default value: False

`mur`

Sets the *maximum unambiguous range* (MUR), which in turn sets the *maximum measurable distance* (MMD).

The MMD is the maximum value for the range end, i.e., the range start + length. The MMD is smaller than the MUR due to hardware limitations.

The MUR is the maximum distance at which an object can be located to guarantee that its reflection corresponds to the most recent transmitted pulse. Objects farther away than the MUR may fold into the measured range. For example, with a MUR of 10 m, an object at 12 m could become visible at 2 m.

A higher setting gives a larger MUR/MMD, but comes at a cost of increasing the measurement time for a sweep. The measurement time is approximately proportional to the MUR.

This setting changes the *pulse repetition frequency* (PRF) of the radar system. The relation between PRF and MUR is

$$\mathrm{MUR} = c/(2 * \mathrm{PRF})$$

where c is the speed of light.

Setting	MUR	MMD	PRF
6	11.5 m	7.0 m	13.0 MHz
9	17.3 m	12.7 m	8.7 MHz

This is an experimental feature.

Default value: MUR.MUR_6

`noise_level_normalization`

With the SW version 2 release, a sensor signal normalization functionality is activated by default for the Power Bins, Envelope, and IQ Service. This results in a more constant signal for different temperatures and sensors. The radar sweeps are normalized to have similar amplitude independent of sensor gain and hardware averaging, resulting in only minor visible effect in the sweeps when adjusting these parameters.

We recommend this setting especially for applications, where absolute radar amplitudes are important, such as when comparing to a previously recorded signal or to a fixed threshold.

More technically, the functionality is implemented to collect data when starting the service, but not transmitting pulses. This data is then used to determine the current sensitivity of receiving part of the radar by estimating the power level of the noise, which then is used to normalize the collected sweeps. In the most low-power systems, where a service is created to collect just a single short sweep before turning off, the sensor normalization can add a non-negligible part to the power consumption.

Please note, that due to the nature of Sparse data, the Sparse service does not support noise level normalization. Instead, normalization during processing is recommended, such as done in the Presence detector.

Type: bool
Default value: True

`power_save_mode`

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes. The modes differentiate in current dissipation and response latency, where the most current consuming mode *Active* gives fastest response and the least current consuming mode *Off* gives the slowest response. The absolute response time and also maximum update rate is determined by several factors besides the power save mode configuration.

In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. The power consumption of the system depends on the actual configuration of the application and it is recommended to test both maximum update rate and power consumption when the configuration is decided.

Power save mode	Current consumption	Response time
Off	Lowest	Longest
Hibernate	...	...
Sleep	...	...
Ready	...	...
Active	Highest	Shortest

Note

Hibernation has limited hardware support. It is not supported by the Raspberry Pi EVK:s and XM112.

Default value: PowerSaveMode.ACTIVE

`profile`

The main configuration of all the services are the profiles, numbered 1 to 5. The difference between the profiles is the length of the radar pulse and the way the incoming pulse is sampled. Profiles with low numbers use short pulses while the higher profiles use longer pulses.

Profile 1 is recommended for:

- measuring strong reflectors, to avoid saturation of the received signal
- close range operation (<20 cm), due to the reduced direct leakage

Profile 2 and 3 are recommended for:

- operation at intermediate distances, (20 cm to 1 m)
- where a balance between SNR and depth resolution is acceptable

Profile 4 and 5 are recommended for:

- for Sparse service only
- operation at large distances (>1 m)
- motion or presence detection, where an optimal SNR ratio is preferred over a high resolution distance measurement

The previous profile Maximize Depth Resolution and Maximize SNR are now profile 1 and 2. The previous Direct Leakage Profile is obtained by the use of the Maximize Signal Attenuation parameter.

Default value: Profile.PROFILE_2

`range_interval`

The measured depth range. The start and end values will be rounded to the closest measurement point available.

The the sweep range is limited to 7.0 m for the default “maximum unambiguous range” setting. To change this limitation, increase “maximum unambiguous range”.

Type: float
Unit: m
Default value: [0.18, 0.78]

`repetition_mode`

The RSS supports two different repetition modes. They determine how and when data acquisition occurs. They are:

- **On demand / host driven**: The sensor produces data when requested by the application. Hence, the application is responsible for timing the data acquisition. This is the default mode, and may be used with all power save modes.
- **Streaming / sensor driven**: The sensor produces data at a fixed rate, given by a configurable accurate hardware timer. This mode is recommended if exact timing between updates is required.

The Exploration Tool is capable of setting the update rate also in *on demand (host driven)* mode. Thus, the difference between the modes becomes subtle. This is why *on demand* and *streaming* are called *host driven* and *sensor driven* respectively in Exploration Tool.

Default value: RepetitionMode.HOST_DRIVEN

`running_average_factor`

The time smoothing factor for Envelope sweeps. With the running average factor larger than zero, consecutive sweeps are averaged using an exponential window function. A runnning average factor of 0.0 corresponds to no time filtering of sweeps and close to 1.0 results in more filtering.

Envelope sweep number s returned by RSS, $E_s(r)$, is calculated from the measured sweep, $e_s(r)$, according to

$$E_s(r) = \text{RAF} \cdot E_{s-1}(r) + (1 - \text{RAF}) \cdot e_s(r),$$

where RAF is the running average factor.

Type: float
Default value: 0.7

`sensor`

The sensor(s) to be configured.

Default value: [1]

`tx_disable`

Disable the radio transmitter. If used to measure noise, we recommended also switching off noise level normalization (if applicable).

Type: bool
Default value: False

`update_rate`

The rate f_f at which the sensor sends frames to the host MCU.

Attention

Setting the update rate too high might result in missed data frames.

In sparse, the maximum possible update rate depends on the *sweeps per frame* N_s and *sweep rate* f_s:

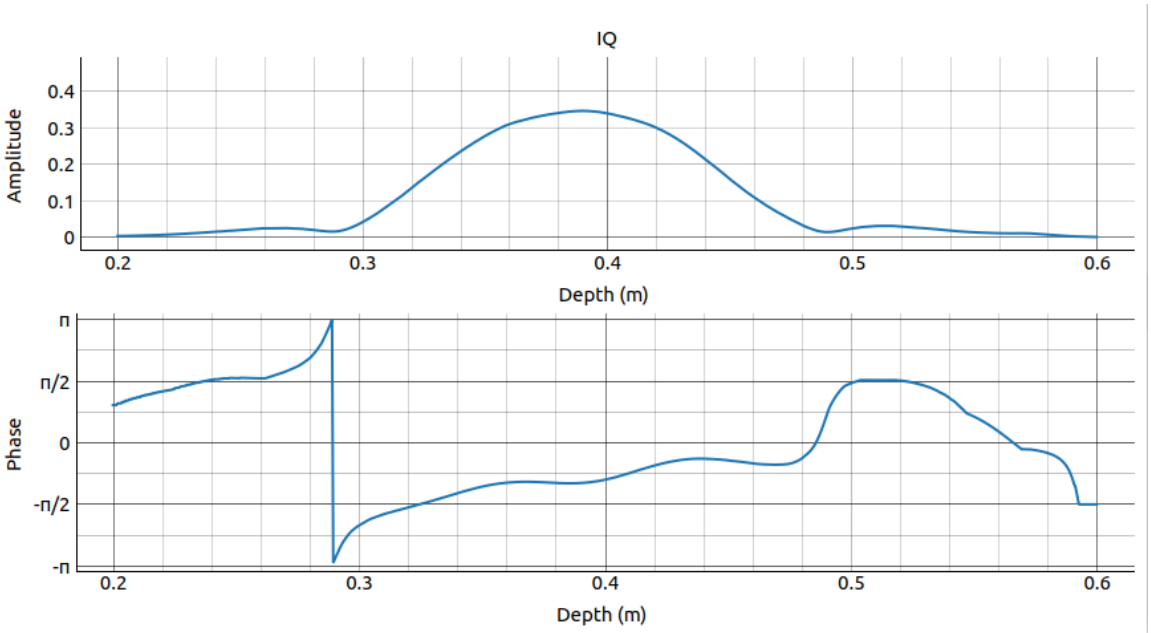
$$\frac{1}{f_f} > N_s \cdot \frac{1}{f_s} + \text{overhead*}$$

* *The overhead largely depends on data frame size and data transfer speeds.*

Type: float
Unit: Hz
Default value: None

19.3 IQ

The IQ service utilizes the phase-coherency of the Acconeer pulsed radar to produce stable In-phase and Quadrature (IQ) components, capable of detecting fine movement occurring in a target scene. Such micro-motions can be used in, for instance, presence detection in front of the sensor, detection of breathing rate and obstacle detection.



The In-phase and Quadrature components are represented as complex values in cartesian form, generating a complex set of N_D samples represented as $x[d]$, where d is the delay sample index. Each complex value has an amplitude and a phase as in $x[d] = A_d e^{i\phi_d}$. A 2π phase rotation of an IQ data point corresponds to a movement at the specific distance of $\lambda/2 \approx 2.5$ mm, providing high relative spatial resolution.

The cartesian data can be transformed to polar data providing phase and amplitude of the signal. Having the phase of the signal available makes it possible to perform more accurate measurements as compared to the Power bins and Envelope Services where only the amplitude is available. This is illustrated in Figure 25 where an object is moving towards the radar. The envelope of the signal only varies slightly when the object is moving, while the value of the coherent signal at a fixed time delay varies substantially. This change will be present in the phase of the data from the IQ Service.

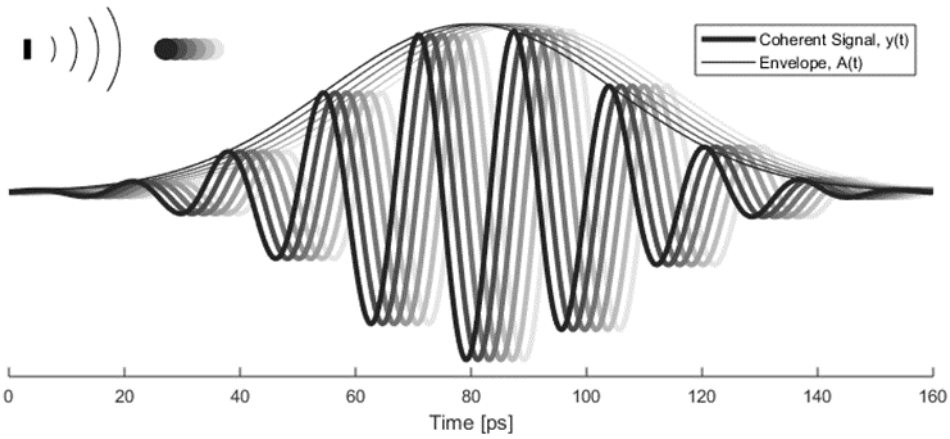


Figure 25: Illustration of envelope and phase change of a received pulse for a reflection from a moving object, what is returned from the IQ Service is in cartesian form.

Similarly to the *Envelope* service the amplitudes obtained through the IQ service provide a method for examining the reflectivity at different distances from the radar sensor. These two services are however differently optimized. The *Envelope* service is optimized for providing an accurate envelope estimate, while the IQ service is optimized for producing a phase-stable estimate. Thus, one should only use the IQ service if phase information is of importance.

For phase estimation in the vital sign use case and for object detection in the robot use case, Profile 2 and 3 are recommended. To get data good data from the IQ service with Profile 1, sampling mode B and a Hardware accelerated average samples (HWAAS) of at least 20.

The filtering of data in the IQ Service applies a low-pass filter in the range dimension. This leads to some filter edge effects in the first few centimeters of the sweep. For very short sweeps, approx. 3 cm for Profile 1 and approx 6 cm for Profile 2-5, these edge effects affects the magnitude and phase of the whole sweep. It is therefore recommended to add at least 3 cm to the sweep at each end for Profile 1, and 6 cm for Profile 2-5, to the region where the amplitude and phase should be estimated.

The IQ service can be configured with different pulse length profiles, see *Profiles*.

Detection of micro-motions using the IQ service in a target scene has many use cases, some of which are presented in sleep-breathing, obstacle-detection, and phase-tracking.

`examples/a111/services/iq.py` contains example code on how the IQ service can be used.

For further reading on the IQ service we refer to the IQ documentation on the Acconeer developer page.

Configuration parameters

`class acconeer.exptool.a111.IQServiceConfig`

`class PowerSaveMode(`*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

`class SamplingMode(`*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

`asynchronous_measurement`

Enabling asynchronous measurements will result in a faster update rate but introduces a risk of interference between sensors.

Type: bool
Default value: True

`depth_lowpass_cutoff_ratio`

Depth domain lowpass filter cutoff frequency ratio

The cutoff for the depth domain lowpass filter is specified as the ratio between the spatial frequency cutoff and the sample frequency. A ratio of zero ratio will configure the smoothest possible filter. A ratio of 0.5 (the Nyquist frequency) turns the filter off.

If unset, i.e., if not overridden, the ratio will be chosen automatically. The used ratio is returned in the session information (metadata) upon session setup (create).

Type: float
Default value: None

`downsampling_factor`

The range downsampling by an integer factor. A factor of 1 means no downsampling, thus sampling with the smallest possible depth interval. A factor 2 samples every other point, and so on. In Envelope and IQ, the finest interval is ~0.5 mm. In Power Bins, it is the same but then further downsampled in post-processing. In sparse, it is ~6 cm.

The downsampling is performed by skipping measurements in the sensor, and therefore gives lower memory usage, lower power consumption, and lower duty cycle.

In sparse, setting a too large factor might result in gaps in the data where moving objects "disappear" between sampling points.

In Envelope, IQ, and Power Bins, the factor must be 1, 2, or 4. In sparse, it must be at least 1. Setting a factor greater than 1 might affect the range end point and for IQ and Envelope, also the first point.

Type: int
Default value: 1

`gain`

The receiver gain used in the sensor. If the gain is too low, objects may not be visible, or it may result in poor signal quality due to quantization errors. If the gain is too high, strong reflections may result in saturated data. We recommend not setting the gain higher than necessary due to signal quality reasons.

Must be between 0 and 1 inclusive, where 1 is the highest possible gain.

> **Note**
>
> When Sensor normalization is active, the change in the data due to changing gain is removed after normalization. Therefore, the data might seen unaffected by changes in the gain, except very high (receiver saturation) or very low (quantization error) gain.
>
> Sensor normalization is not available for the Sparse service, but is enabled by default for the other services - Envelope, IQ, and Power Bins.

Type: float
Default value: 0.5

`hw_accelerated_average_samples`

Number of samples taken to obtain a single point in the data. These are averaged directly in the sensor hardware - no extra computations are done in the MCU.

The time needed to measure a sweep is roughly proportional to the HWAAS. Hence, if there's a need to obtain a higher sweep rate, HWAAS could be decreased. Note that HWAAS does not affect the amount of data transmitted from the sensor over SPI.

Must be at least 1 and not greater than 63.

Type: int
Default value: 10

`maximize_signal_attenuation`

When measuring in the direct leakage (around 0m), this setting can be enabled to minimize saturation in the receiver. We do not recommend using this setting under normal operation.

Type: bool
Default value: False

`noise_level_normalization`

With the SW version 2 release, a sensor signal normalization functionality is activated by default for the Power Bins, Envelope, and IQ Service. This results in a more constant signal for different temperatures and sensors. The radar sweeps are normalized to have similar amplitude independent of sensor gain and hardware averaging, resulting in only minor visible effect in the sweeps when adjusting these parameters.

We recommend this setting especially for applications, where absolute radar amplitudes are important, such as when comparing to a previously recorded signal or to a fixed threshold.

More technically, the functionality is implemented to collect data when starting the service, but not transmitting pulses. This data is then used to determine the current sensitivity of receiving part of the radar by estimating the power level of the noise, which then is used to normalize the collected sweeps. In the most low-power systems, where a service is created to collect just a single short sweep before turning off, the sensor normalization can add a non-negligible part to the power consumption.

Please note, that due to the nature of Sparse data, the Sparse service does not support noise level normalization. Instead, normalization during processing is recommended, such as done in the Presence detector.

Type: bool
Default value: True

`power_save_mode`

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes. The modes differentiate in current dissipation and response latency, where the most current consuming mode *Active* gives fastest response and the least current consuming mode *Off* gives the slowest response. The absolute response time and also maximum update rate is determined by several factors besides the power save mode configuration.

In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. The power consumption of the system depends on the actual configuration of the application and it is recommended to test both maximum update rate and power consumption when the configuration is decided.

Power save mode	Current consumption	Response time
Off	Lowest	Longest
Hibernate	...	...
Sleep	...	...
Ready	...	...
Active	Highest	Shortest

Note

Hibernation has limited hardware support. It is not supported by the Raspberry Pi EVK:s and XM112.

Default value: PowerSaveMode.ACTIVE

`profile`

The main configuration of all the services are the profiles, numbered 1 to 5. The difference between the profiles is the length of the radar pulse and the way the incoming pulse is sampled. Profiles with low numbers use short pulses while the higher profiles use longer pulses.

Profile 1 is recommended for:

- measuring strong reflectors, to avoid saturation of the received signal
- close range operation (<20 cm), due to the reduced direct leakage

Profile 2 and 3 are recommended for:

- operation at intermediate distances, (20 cm to 1 m)
- where a balance between SNR and depth resolution is acceptable

Profile 4 and 5 are recommended for:

- for Sparse service only
- operation at large distances (>1 m)
- motion or presence detection, where an optimal SNR ratio is preferred over a high resolution distance measurement

The previous profile Maximize Depth Resolution and Maximize SNR are now profile 1 and 2. The previous Direct Leakage Profile is obtained by the use of the Maximize Signal Attenuation parameter.

Default value: Profile.PROFILE_2

`range_interval`

The measured depth range. The start and end values will be rounded to the closest measurement point available.

Type: float
Unit: m
Default value: [0.18, 0.78]

`repetition_mode`

The RSS supports two different repetition modes. They determine how and when data acquisition occurs. They are:

- **On demand / host driven**: The sensor produces data when requested by the application. Hence, the application is responsible for timing the data acquisition. This is the default mode, and may be used with all power save modes.
- **Streaming / sensor driven**: The sensor produces data at a fixed rate, given by a configurable accurate hardware timer. This mode is recommended if exact timing between updates is required.

The Exploration Tool is capable of setting the update rate also in *on demand (host driven)* mode. Thus, the difference between the modes becomes subtle. This is why *on demand* and *streaming* are called *host driven* and *sensor driven* respectively in Exploration Tool.

Default value: RepetitionMode.HOST_DRIVEN

`sampling_mode`

Default value: SamplingMode.A

`sensor`
: The sensor(s) to be configured.

  Default value: [1]

`tx_disable`
: Disable the radio transmitter. If used to measure noise, we recommended also switching off noise level normalization (if applicable).

  Type: bool
  Default value: False

`update_rate`
: The rate f_f at which the sensor sends frames to the host MCU.

  Attention

  Setting the update rate too high might result in missed data frames.

  In sparse, the maximum possible update rate depends on the *sweeps per frame* N_s and *sweep rate* f_s:

$$\frac{1}{f_f} > N_s \cdot \frac{1}{f_s} + \text{overhead*}$$

  * *The overhead largely depends on data frame size and data transfer speeds.*

  Type: float
  Unit: Hz
  Default value: None

19.4 Sparse

The sparse service is ideal for motion-sensing applications requiring high robustness and low power consumption.

`examples/a111/services/sparse.py` contains example code on how the Sparse service can be used.

How it works

The other services, *Envelope*, *IQ*, and *Power Bins*, are all based on sampling the incoming waves several times per wavelength (effectively ~2.5 mm). In sparse, the incoming waves are instead sampled every ~6 **cm**. Therefore, sparse is fundamentally different from the other services. It **does not** simply produce a downsampled version of the *Envelope* or *IQ* service.

Due to the highly undersampled signal from the sparse service, it should not be used to measure the reflections of static objects. Instead, the sparse service should be used for situations where detecting moving objects is desired. Sparse is optimal for this, as it produces sequences of very robust measurements at the sparsely located sampling points.

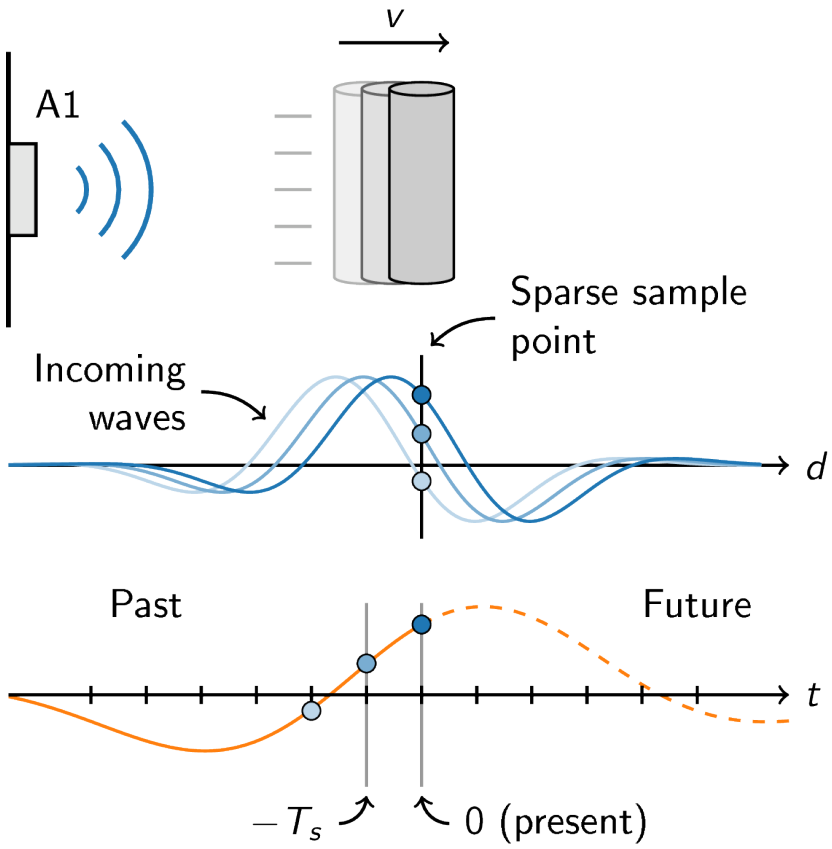


Figure 26: An illustration of how the sparse service samples reflected waves from a moving object.

The above image illustrates how a single sparse point samples incoming wavelets from a moving target. The three different blue colored waves are from different points in time, where the darkest one is the most recent (present), and the faded ones are from the past. For every point in time, a sample is taken at the sampling point(s). In this example, there is only one single sample point, but in reality, most often several points are used.

The bottom plot lays out the sampled points over a time scale. In this simple example, the object moves with a steady velocity. As such, over time, the samples will reconstruct the incoming wavelet, which the orange line illustrates.

Note

If the target object is static, the signal too will be static, but not necessarily zero. It can take any value within the peak values of the reflected wave, depending on where on the wave it is sampled.

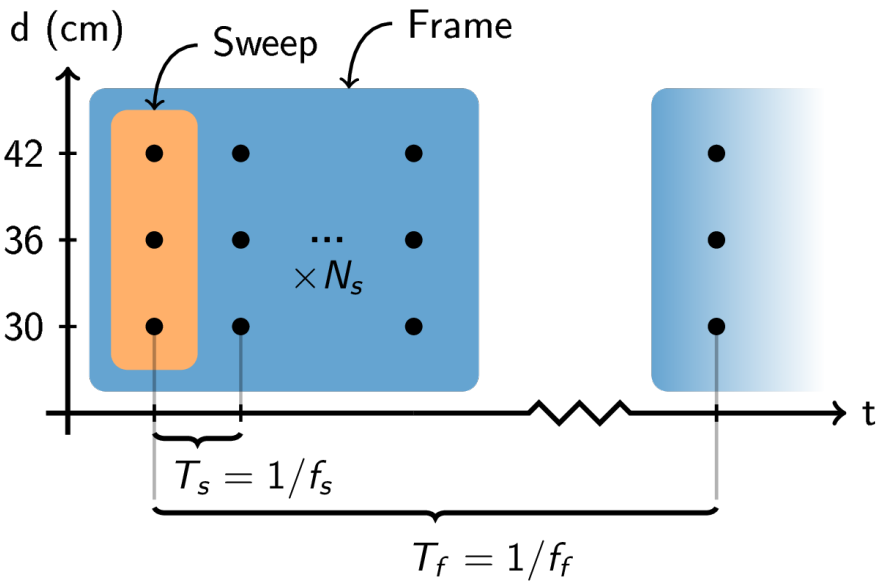


Figure 27: An illustration of the sparse data frames consisting of a number of sweeps.

From sparse, every received data frame consists of a number of sweeps N_s which are sampled after each other. Every sweep consists of one or several (sparse) sampling points in distance as configured. Depending on the configuration, the time between sweeps T_s may vary. It can also, within certain limits, be set to a fixed value. Often, we refer to this as the *sweep rate* $f_s = 1/T_s$ instead of referring to the time between sweeps T_s.

Typical sweep rates range between 1 and 50 kHz. On the other hand, typical frame (update) rates f_f range between 1 and 200 Hz. Therefore, there often is a large gap between the end of a frame to the beginning of the next one. From a power consumption perspective, this is desirable since it allows the sensor to have a smaller duty cycle. However, if needed, the sparse service can be configured for a near 100% duty cycle.

For many applications, sweeps are sampled closely enough in time that they can be regarded as being sampled simultaneously. In such applications, for example presence detection, the sweeps in each frame can be averaged for optimal SNR.

Tip

Unlike the other services, there is no processing applied to the radar data. Also, it typically produces less data. This makes the sparse service relatively computationally inexpensive and suitable for use with smaller MCU:s.

The sparse service utilizes longer wavelets than the other services, meaning that there will be more energy and therefore better SNR in the received signal. For example, this results in an increased distance coverage for presence detection applications. This also means that a wavelet often spans several sparse points.

How to use

Tip

If this is your first time working with the sparse service, we recommend first getting a feel for how it can be used by running and looking at the presence detector.

While the sparse service has a lot of configuration options, they all have sensible defaults for most applications. The only parameters that you really need to set up to get started is the *range* and *frame (update) rate*. Other parameters can be tuned as you go.

If you're doing things like **gesture recognition** or any **velocity measurements**, we recommend using sampling mode A and explicitly setting the *sweep rate*. Also, raising the number of sweeps per frame might be beneficial for such measurements. From there it is also often a good idea to set the *frame (update) rate* as high as possible. Lowering the number of *HWAAS* might be necessary to obtain the desired sweep and/or frame rate.

Configuration parameters

`class acconeer.exptool.a111.SparseServiceConfig`

`class MUR(value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None)`

`class PowerSaveMode(value, names=<not given>, *values, module=None, qualname=None, type=None, start=1, boundary=None)`

`asynchronous_measurement`

Enabling asynchronous measurements will result in a faster update rate but introduces a risk of interference between sensors.

Type: bool
Default value: True

`downsampling_factor`

The range downsampling by an integer factor. A factor of 1 means no downsampling, thus sampling with the smallest possible depth interval. A factor 2 samples every other point, and so on. In Envelope and IQ, the finest interval is ~0.5 mm. In Power Bins, it is the same but then further downsampled in post-processing. In sparse, it is ~6 cm.

The downsampling is performed by skipping measurements in the sensor, and therefore gives lower memory usage, lower power consumption, and lower duty cycle.

In sparse, setting a too large factor might result in gaps in the data where moving objects "disappear" between sampling points.

In Envelope, IQ, and Power Bins, the factor must be 1, 2, or 4. In sparse, it must be at least 1. Setting a factor greater than 1 might affect the range end point and for IQ and Envelope, also the first point.

Type: int
Default value: 1

`gain`

The receiver gain used in the sensor. If the gain is too low, objects may not be visible, or it may result in poor signal quality due to quantization errors. If the gain is too high, strong reflections may result in saturated data. We recommend not setting the gain higher than necessary due to signal quality reasons.

Must be between 0 and 1 inclusive, where 1 is the highest possible gain.

> **Note**
>
> When Sensor normalization is active, the change in the data due to changing gain is removed after normalization. Therefore, the data might seen unaffected by changes in the gain, except very high (receiver saturation) or very low (quantization error) gain.
>
> Sensor normalization is not available for the Sparse service, but is enabled by default for the other services - Envelope, IQ, and Power Bins.

Type: float
Default value: 0.5

`hw_accelerated_average_samples`

Number of samples taken to obtain a single point in the data. These are averaged directly in the sensor hardware - no extra computations are done in the MCU.

The time needed to measure a sweep is roughly proportional to the HWAAS. Hence, if there's a need to obtain a higher sweep rate, HWAAS could be decreased. Note that HWAAS does not affect the amount of data transmitted from the sensor over SPI.

Must be at least 1 and not greater than 63.

Type: int
Default value: 10

`maximize_signal_attenuation`

When measuring in the direct leakage (around 0m), this setting can be enabled to minimize saturation in the receiver. We do not recommend using this setting under normal operation.

Type: bool
Default value: False

`mur`

Sets the *maximum unambiguous range* (MUR), which in turn sets the *maximum measurable distance* (MMD).

The MMD is the maximum value for the range end, i.e., the range start + length. The MMD is smaller than the MUR due to hardware limitations.

The MUR is the maximum distance at which an object can be located to guarantee that its reflection corresponds to the most recent transmitted pulse. Objects farther away than the MUR may fold into the measured range. For example, with a MUR of 10 m, an object at 12 m could become visible at 2 m.

A higher setting gives a larger MUR/MMD, but comes at a cost of increasing the measurement time for a sweep. The measurement time is approximately proportional to the MUR.

This setting changes the *pulse repetition frequency* (PRF) of the radar system. The relation between PRF and MUR is

$$\mathrm{MUR} = c/(2 * \mathrm{PRF})$$

where c is the speed of light.

Setting	MUR	MMD	PRF
6	11.5 m	7.0 m	13.0 MHz
9	17.3 m	12.7 m	8.7 MHz

This is an experimental feature.

Default value: MUR.MUR_6

`power_save_mode`

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes. The modes differentiate in current dissipation and response latency, where the most current consuming mode *Active* gives fastest response and the least current consuming mode *Off* gives the slowest response. The absolute response time and also maximum update rate is determined by several factors besides the power save mode configuration.

In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. The power consumption of the system depends on the actual configuration of the application and it is recommended to test both maximum update rate and power consumption when the configuration is decided.

Power save mode	Current consumption	Response time
Off	Lowest	Longest
Hibernate	...	...
Sleep	...	...
Ready	...	...
Active	Highest	Shortest

Note

Hibernation has limited hardware support. It is not supported by the Raspberry Pi EVK:s and XM112.

Default value: PowerSaveMode.ACTIVE

`profile`

The main configuration of all the services are the profiles, numbered 1 to 5. The difference between the profiles is the length of the radar pulse and the way the incoming pulse is sampled. Profiles with low numbers use short pulses while the higher profiles use longer pulses.

Profile 1 is recommended for:

- measuring strong reflectors, to avoid saturation of the received signal
- close range operation (<20 cm), due to the reduced direct leakage

Profile 2 and 3 are recommended for:

- operation at intermediate distances, (20 cm to 1 m)
- where a balance between SNR and depth resolution is acceptable

Profile 4 and 5 are recommended for:

- for Sparse service only
- operation at large distances (>1 m)
- motion or presence detection, where an optimal SNR ratio is preferred over a high resolution distance measurement

The previous profile Maximize Depth Resolution and Maximize SNR are now profile 1 and 2. The previous Direct Leakage Profile is obtained by the use of the Maximize Signal Attenuation parameter.

Default value: Profile.PROFILE_2

`range_interval`

The measured depth range. The start and end values will be rounded to the closest measurement point available.

The the sweep range is limited to 7.0 m for the default “maximum unambiguous range” setting. To change this limitation, increase “maximum unambiguous range”.

Type: float
Unit: m
Default value: [0.18, 0.78]

`repetition_mode`

The RSS supports two different repetition modes. They determine how and when data acquisition occurs. They are:

- **On demand / host driven**: The sensor produces data when requested by the application. Hence, the application is responsible for timing the data acquisition. This is the default mode, and may be used with all power save modes.
- **Streaming / sensor driven**: The sensor produces data at a fixed rate, given by a configurable accurate hardware timer. This mode is recommended if exact timing between updates is required.

The Exploration Tool is capable of setting the update rate also in *on demand (host driven)* mode. Thus, the difference between the modes becomes subtle. This is why *on demand* and *streaming* are called *host driven* and *sensor driven* respectively in Exploration Tool.

Default value: RepetitionMode.HOST_DRIVEN

`sampling_mode`

The sampling mode changes how the hardware accelerated averaging is done. This may either increase SNR or reduce correlation.

Mode A is:

- optimized for maximal independence of the depth points, giving a higher depth resolution than mode B.
- more suitable for applications like gesture recognition, measuring the distance to a movement, and speed measurements.

Mode B is:

- optimized for maximal SNR per unit time spent on measuring. This makes it more energy efficient and suitable for cases where small movements are to be detected over long ranges.
- resulting in roughly 3 dB better SNR per unit time than mode A.

Default value: SamplingMode.B

`sensor`

The sensor(s) to be configured.

Default value: [1]

`sweep_rate`

In Sparse, each frame is a collection of several sweeps over the selected distance range (sweeps per frame). The sweep rate f_s is the rate at which sweeps are performed, i.e. the rate at which each distance point is scanned. If you set the sweep rate to 4000 Hz and the sweeps per frame to 32, each Sparse data frame will contain 32 sweeps over the selected distance range, where the sweeps are measured at a rate of 4000 Hz.

The maximum possible sweep rate...

- Is roughly inversely proportional to the number of depth points measured (affected by the **range interval** and **downsampling factor**).
- Is roughly inversely proportional to **HW accelerated average samples**.
- Depends on the **sampling mode**. Mode A is roughly $4/3 \approx 130\%$ slower than mode B with the same configuration.

To get the maximum possible rate, leave this value unset and look at the *sweep rate* in the session info (metadata).

Tip

If you do not need a specific sweep rate, we recommend leaving it unset.

Type: float
Unit: Hz
Default value: None

`sweeps_per_frame`
: The number of sweeps per frame N_s.

Must be at least 1, and not greater than 64 when using sampling mode B.

Type: int
Default value: 16

`tx_disable`
: Disable the radio transmitter. If used to measure noise, we recommended also switching off noise level normalization (if applicable).

Type: bool
Default value: False

`update_rate`
: The rate f_f at which the sensor sends frames to the host MCU.

> **Attention**
>
> Setting the update rate too high might result in missed data frames.

In sparse, the maximum possible update rate depends on the *sweeps per frame* N_s and *sweep rate* f_s:

$$\frac{1}{f_f} > N_s \cdot \frac{1}{f_s} + \text{overhead*}$$

* *The overhead largely depends on data frame size and data transfer speeds.*

Type: float
Unit: Hz
Default value: None

Session info (metadata)

The following information is returned when creating a session.

Data length

Python: `data_length`
C SDK: `data_length`
Type: int

The size of the data frames. For sparse, this is the number of depths times the number of sweeps.

Range start

Python: `range_start_m`
C SDK: `start_m`
Type: float
Unit: m

The depth of range start - after rounding the configured range to the closest available sparse sampling points.

Range length

Python: `range_length_m`
C SDK: `length_m`
Type: float
Unit: m

The length of the range - after rounding the configured range to the closest available sparse sampling points.

Step length

Python: `step_length_m`
C SDK: `step_length_m`
Type: float
Unit: m

The distance in meters between points in the range.

Sweep rate

Python: `sweep_rate`
C SDK: `sweep_rate`
Type: float
Unit: Hz

The sweep rate. If a sweep rate is explicitly set, this value will be very close to the configured value. If not (the default), this will be the maximum sweep rate.

Data info (result info)

The following information is returned with each data frame.

Missed data

Python and C SDK: `missed_data`
Type: int

Indicates if a data frame was missed, for example due to a too high `update_rate`.

Data saturated

Python and C SDK: `data_saturated`
Type: bool

Indication that the sensor has hit its full dynamic range. If this indication is given, the result might be unstable. Most often, the problem is that the gain is set too high.

Disclaimer

The sparse service will have optimal performance using any one of the XM112, XM122 or XM132 Modules. A111 with batch number 10467, 10457 or 10178 (also when mounted on XR111 and XR112) should be avoided when using the sparse service.

20 Detectors

Detectors take Service data as input and produce a result as the output that can be used by the application. Currently we have four Detectors available that produce different types of results and that are based on different Services. User guides for the different Detectors are available at acconeer.com and the Detectors are also available in the Exploration Tool.

In addition, we provide several Reference applications which use Services or Detectors to demonstrate how to develop applications based on our technology, you can find these in the various SDKs at Acconeer developer site.

20.1 Distance detector

This is a distance detector algorithm built on top of the *Envelope* service – based on comparing the envelope sweep to a threshold and identifying one or more peaks in the envelope sweep, corresponding to objects in front of the radar. The algorithm both detects the presence of objects and estimates their distance to the radar. More details about the detector is found here.

20.2 Presence detector

Detects changes in the environment over time based on data from the Sparse service. More details about the detector is found here.

20.3 Obstacle detector

Assumes that the Acconeer sensor is placed on a moving object with a known velocity, such as a robotic vacuum cleaner or lawn mower. The detector creates a virtual antenna array and uses synthetic aperture radar (SAR) signal processing to localize objects. This detector is used in the Obstacle localization demo movie. More details about the detector is found here.

21 System overview

The Acconeer sensor is a mm wavelength pulsed coherent radar, which means that it transmits radio signals in short pulses where the starting phase is well known, as illustrated in Figure 28.

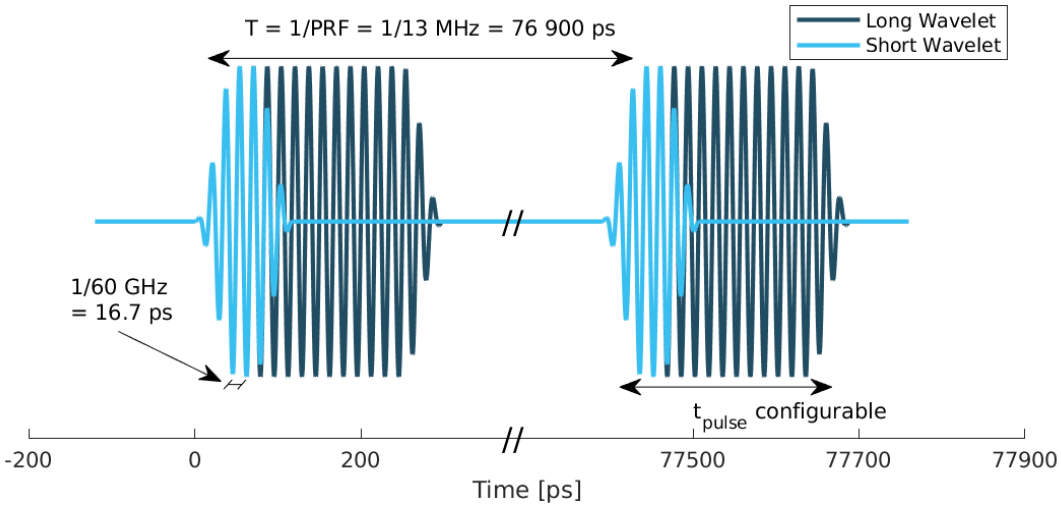


Figure 28: Illustration of the time domain transmitted signal from the Acconeer A111 sensor, a radar sweep typically consists of thousands of pulses. The length of the pulses can be controlled by setting Profile.

These transmitted signals are reflected by an object and the time elapsed between transmission and reception of the reflected signal (t_{delay}) is used to calculate the distance to the object by using

$$d = \frac{t_{delay} v}{2} \tag{14}$$

$$v = \frac{c_0}{\sqrt{\varepsilon_r}} \tag{15}$$

where ε_r is the relative permittivity of the medium. The '2' in the denominator of Eq. 14 is due to the fact that t_{delay} is the time for the signal to travel to the object and back, hence to get the distance to the object a division by 2 is needed. The wavelength λ of the 60.5 GHz carrier frequency f_{RF} is roughly 5 mm in free space. This means that a 5 mm shift of the received wavelet corresponds to a 2.5 mm shift of the detected object due to the round trip distance.

Figure 29 shows a block diagram of the A111 sensor. The signal is transmitted from the Tx antenna and received by the Rx antenna, both integrated in the top layer of the A111 package substrate. In addition to the mmWave radio the sensor consists of power management and digital control, signal quantization, memory and a timing circuit.

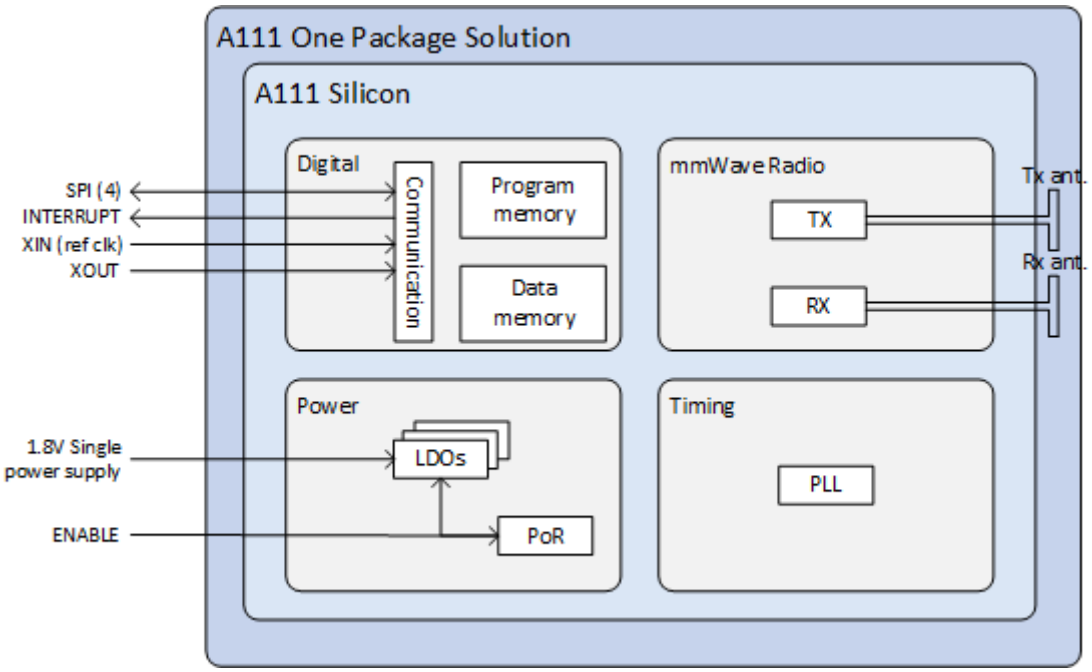


Figure 29: Block diagram of the A111 sensor package, further details about interfaces can be found in the A111 data sheet.

Figure 30 shows a typical radar sweep obtained with the Envelope Service, with one object present. The range resolution of the measurement is ~0.5 mm and each data point correspond to transmission of at least one pulse (depending on averaging), hence, to sweep 30 cm, e.g. from 20 cm to 50 cm as in Figure 30, requires that 600 pulses are transmitted. The system relies on the fact that the pulses are transmitted phase coherent, which makes it possible to send multiple pulses and then combine the received signal from these pulses to improve signal-to-noise ratio (SNR) to enhance the object visibility.

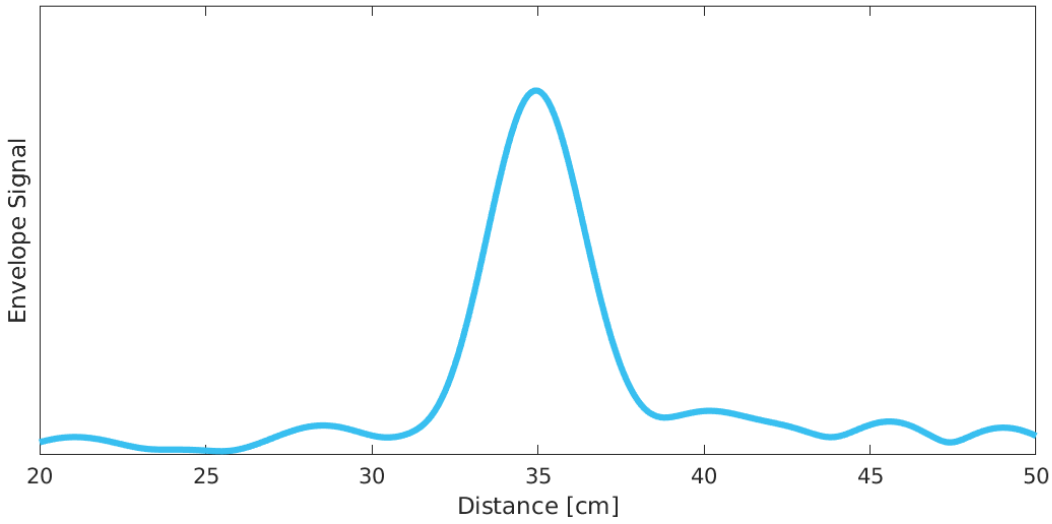


Figure 30: Output from Envelope service for a typical radar sweep with one object present.

22 Performance metrics

Radar sensor performance metrics (RSPMs) for the Acconeer radar system provides useful information on the performance of the system: sensor, RSS and reference integration. The list contains the RSPMs that are applicable to services that produce radar data. However, not all RSPMs are applicable to all radar services. The RSPMs is used in our Radar Datasheet.

22.1 Radar loop gain

The SNR can be modelled as a function of a limited number of parameters: the RCS of the object (σ), the distance to the object (R), the reflectivity of the object (γ), and a radar sensor dependent constant referred to as radar loop gain (C). The SNR (in dB) is then given by

$$\mathrm{SNR}_{dB} = 10\log_{10}\frac{S}{N} = C_{dB} + \sigma_{dB} + \gamma_{dB} - k10\log_{10}R \tag{16}$$

Figure 31 shows how the received energy drops with increasing R for objects where the exponent k is equal to 4, which applies for objects which are smaller than the area which is illuminated coherently by the radar. For objects that are larger than this area the k is smaller than 4, with a lower limit of $k = 2$ when the object is a large flat surface.

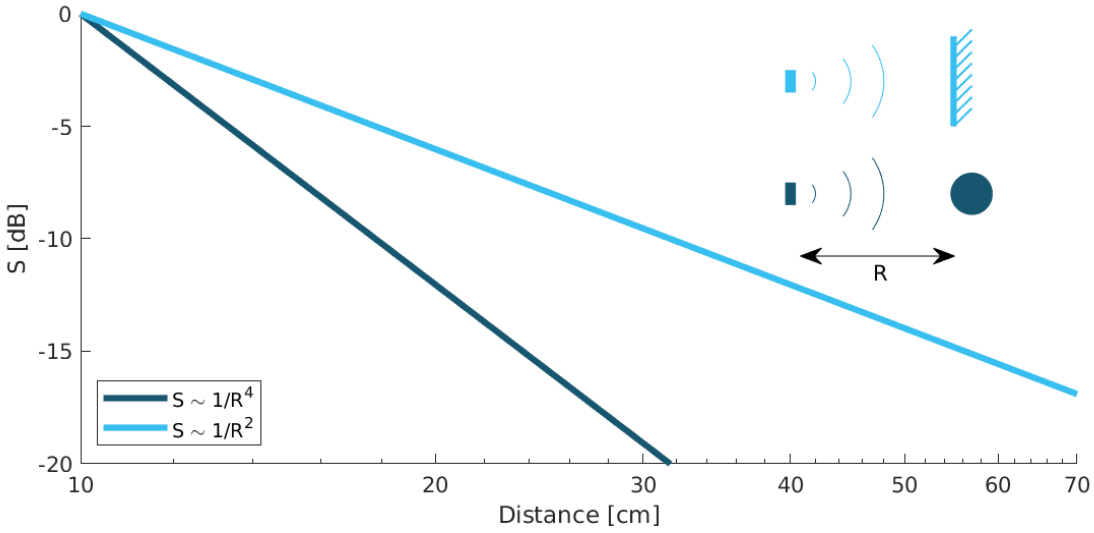


Figure 31: Received signal power versus distance. Note: signal, S, is plotted in dB.

22.2 Depth resolution

The depth resolution determines the minimum distance of two different objects in order to be distinguished from each other.

22.3 Distance resolution

The Acconeer radar systems are based on a time diluted measurement that splits up as a vector of energy in several time bins it is important to know the bin separation. This is the delay resolution of the system and in A111 radar sensor the target is ~3 ps on average, which corresponds to a distance resolution of ~0.5 mm between distance samples.

22.4 Half-power beamwidth

The half-power beamwidth (HPBW) radiation pattern determines the angle between the half-power (-3 dB) points of the main lobe of the radiation pattern. The radiation pattern of the sensor depends on both the antenna-in-package design and the hardware integration of the sensor, such as surrounding components, ground plane size, and added di-electric lenses for directivity optimizations, valid for both vertical and horizontal plane.

22.5 Distance jitter

The distance jitter determines the timing accuracy and stability of the radar system between sweep updates. The jitter is estimated by calculating the standard deviation of the phase, for the same distance bin, over many IQ sweeps.

22.6 Distance linearity

The distance linearity deterministic the deterministic error from the ideal delay transfer function. Linearity of the service data is estimated by measuring the phase change of the IQ data vs distance.

22.7 Update rate accuracy

The update rate accuracy determines the accuracy of the time between sweep updates or similarly the accuracy of the update rate, typically important when the radar data is used for estimating velocity of an object.

22.8 Close-in range

The close-in range determines the radar system limits on how close to the radar sensor objects can be measured.

22.9 Power consumption

The power consumption determines the radar sensor power usage for different configurations as service depends, the power save mode, the update rate, downsampling, sweep length, etc.

Part IV

Appendix

23 Disclaimer

The information herein is believed to be correct as of the date issued. Acconeer AB (“Acconeer”) will not be responsible for damages of any nature resulting from the use or reliance upon the information contained herein. Acconeer makes no warranties, expressed or implied, of merchantability or fitness for a particular purpose or course of performance or usage of trade. Therefore, it is the user’s responsibility to thoroughly test the product in their particular application to determine its performance, efficacy and safety. Users should obtain the latest relevant information before placing orders.

Unless Acconeer has explicitly designated an individual Acconeer product as meeting the requirement of a particular industry standard, Acconeer is not responsible for any failure to meet such industry standard requirements.

Unless explicitly stated herein this document Acconeer has not performed any regulatory conformity test. It is the user’s responsibility to assure that necessary regulatory conditions are met and approvals have been obtained when using the product. Regardless of whether the product has passed any conformity test, this document does not constitute any regulatory approval of the user’s product or application using Acconeer’s product.

Nothing contained herein is to be considered as permission or a recommendation to infringe any patent or any other intellectual property right. No license, express or implied, to any intellectual property right is granted by Acconeer herein.

Acconeer reserves the right to at any time correct, change, amend, enhance, modify, and improve this document and/or Acconeer products without notice.

This document supersedes and replaces all information supplied prior to the publication hereof.