A121 Radar Data and Control

User Guide

A121 Radar Data and Control

User Guide

Author: Acconeer AB

Version:a121-v1.13.0

Acconeer AB March 26, 2026

Contents

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 Vibration Example Application	Describes the functionality of the Vibration Example Application.	- Working with the Vibration Example 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
A121 ESP32 User Guide	Describes how to develop with A121 and ESP32 targets.	- Working with ESP32 targets
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
I2C Cargo Example Application	Describes the functionality of the I2C Cargo Example Application.	- Working with the I2C Cargo Example Application
	A121 Radar Data and Control (PDF)	
A121 Radar Data and Control	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

2 Topics

The topics in this section covers a variety of things from how to interpret the radar data coming out from the sensor to how different configurations affect the radar data.

2.1 API References

For the Python API reference, see Python API reference.

For the C API reference, it can be found in any of our SDKs downloadable from the .

2.2 Differences between A121 and A111

Here are some of the main differences, additions, and highlights with A121 compared to 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.

2.3 How to configure

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?

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 1 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 does not necessarily have to pass r_{near} and r_{far}. Again, see Figure 1 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.

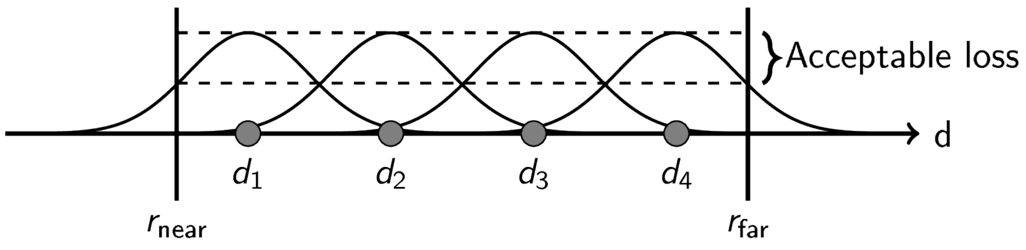

Figure 1: A sketch of setting up the measurement range for efficient coverage of a given area.

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 ($\gtrsim$ 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

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

  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 2 for typical values.

Table 2: 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

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/stabilization 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.

Other parameters

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

2.4 Interpreting radar data

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.

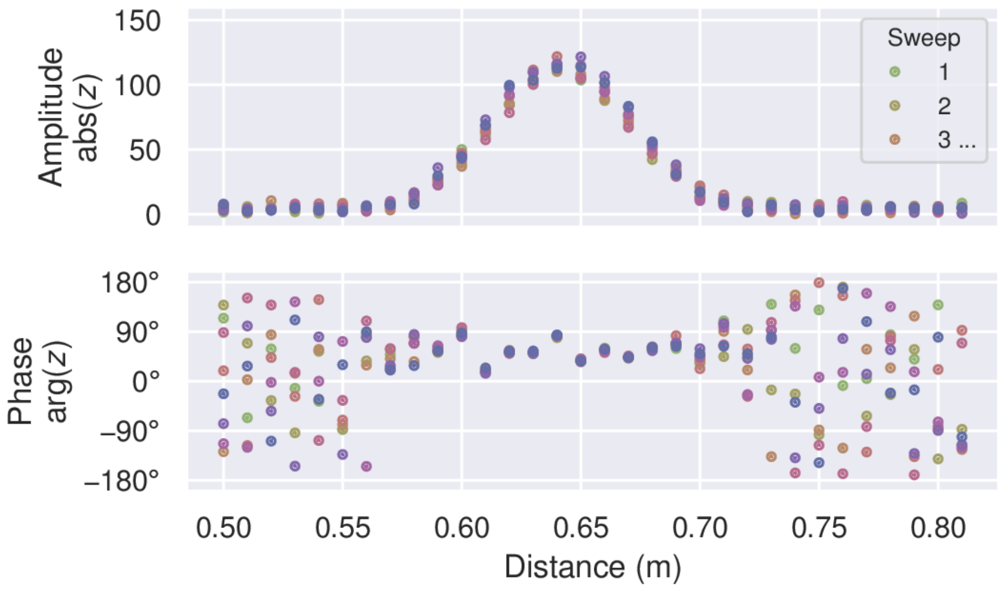

Figure 2: Mocked data of an environment with a single **static** object.

In the simplest case, we have a static environment in the sensor range. Figure 2 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”.

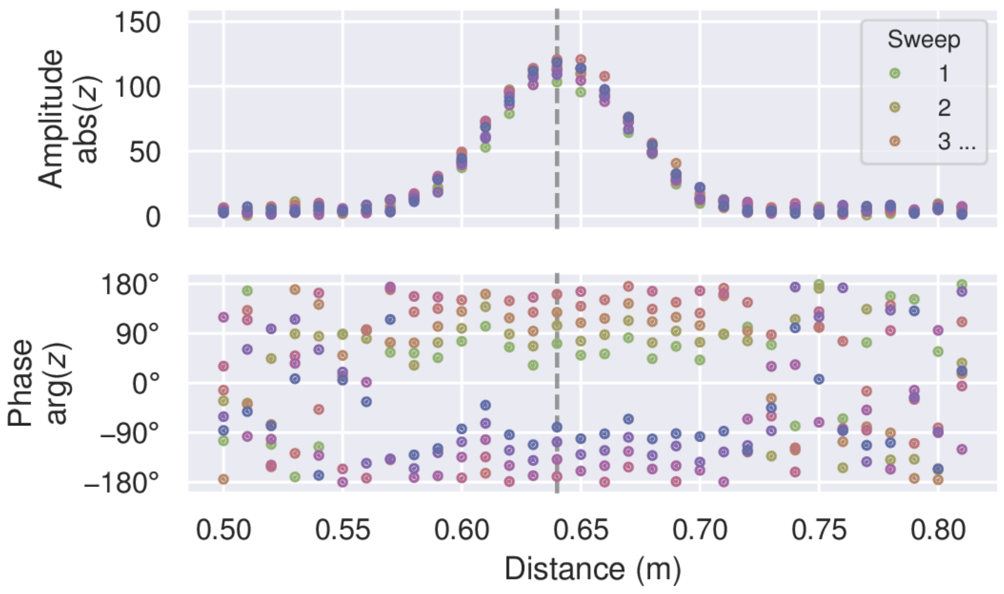

Figure 3: Mocked data of an environment with a single **moving** object.

In many cases, we want to track and/or detect moving objects in the range. This is demonstrated in Figure 3, 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 (non-coherently) average the amplitudes.

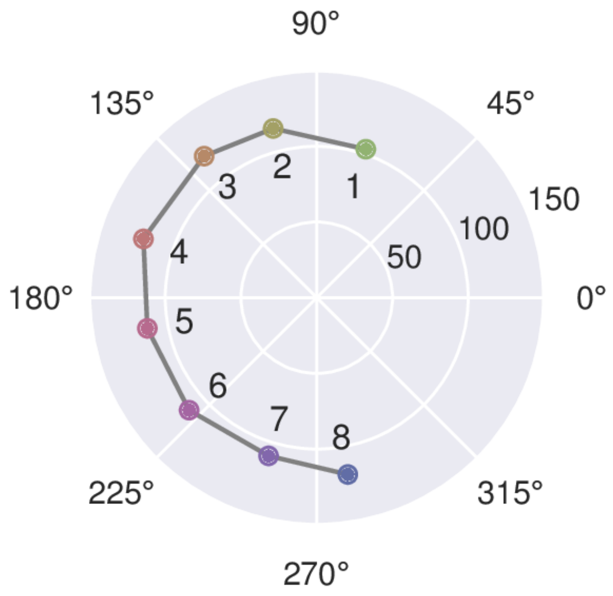

Figure 4: A slice of the mocked data in Figure 3 of an environment with a single moving object, shown in the complex plane.

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 4 shows the slice of the data along the dashed vertical line in Figure 3. Over the 8 sweeps in the example frame, the phase changed ~ 210°. A full phase rotation of 360° translates to $\lambda_{\text{RF}}/2 \approx 2.5$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".

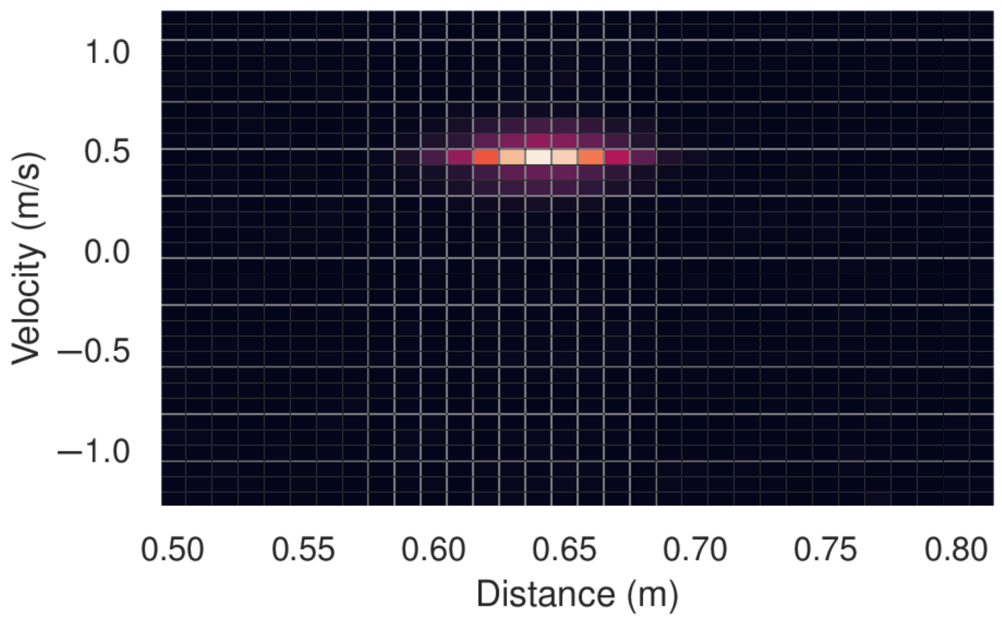

Figure 5: Mocked data of a single moving object, transformed into a distance-velocity (a.k.a. range-Doppler) map.

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 5 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.

2.5 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.

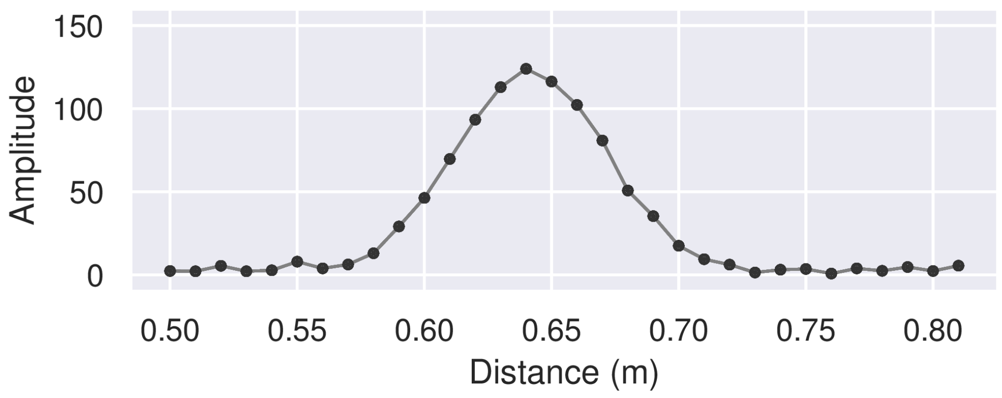

Figure 6: Amplitude of a mocked sweep of an environment with a single object.

Figure 6 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.

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.

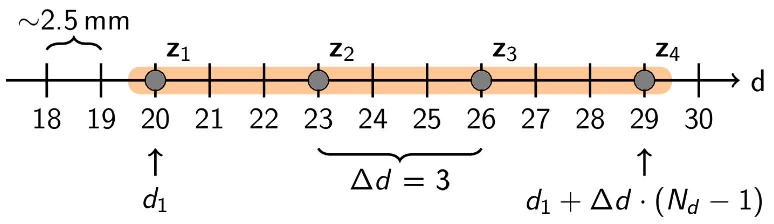

Figure 7: An illustration of the sweep *range* concept.

Figure 7 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.

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 continued	
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.

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”.

2.6 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 *Timing* for a detailed description of the timing within a frame.

2.7 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.

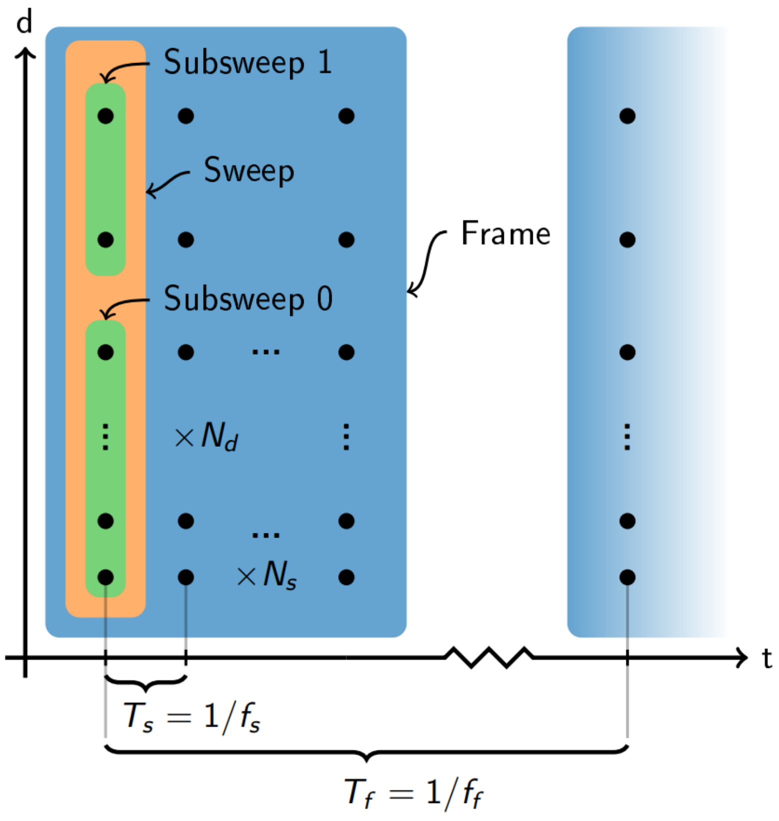

Figure 8: An illustration of the *sweep* and *frame* concept.

As shown in Figure 8, 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.

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.

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.

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, $N_d \cdot N_s \leq 4095$.

2.8 Timing

Note

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

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.

Overview

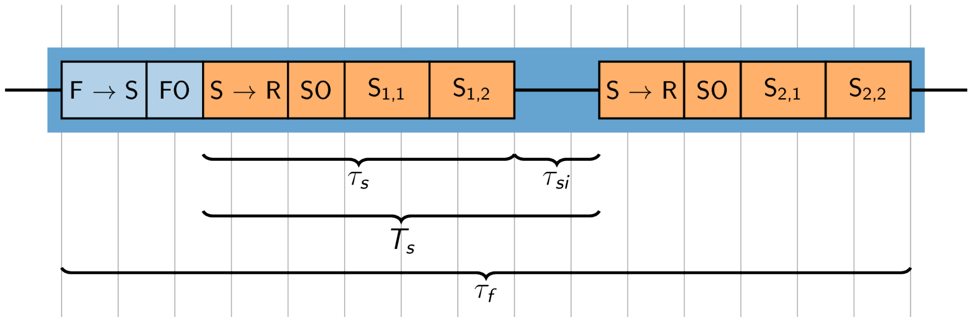

Figure 9: Overview for timing within a frame with two sweeps, in turn consisting of two subsweeps. **Not to scale**, and subsweeps may have different lengths. *F→S* and *S→R* are the times required to transition from the configured *inter frame idle state* to the *inter sweep idle state* state, and *inter 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.

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* according to Table 4 below.

Table 4: 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

Point duration

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

$$\tau_{\text{point}} \approx N_a \cdot \tau_{\text{sample}} + \tau_{\text{point overhead}} \tag{3}$$

where N_a is the configured HWAAS (number of averages), and $\tau_{\text{point overhead}}$ is given by Table 5 below.

Table 5: 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

Subsweep duration

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

$$\tau_{\text{subsweep}} \approx N_d \cdot \tau_{\text{point}} + \underbrace{3 \cdot \tau_{\text{sample}} + C_{\text{subsweep}}}_{\text{overhead}} \tag{4}$$

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)*.

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.

$$\tau_s \approx \tau_{S\to R} + \sum(\tau_{\text{subsweep}}) + C_s \tag{5}$$

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 6 below.

Idle state transition times

Table 6: 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

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}.

$$T_s = \frac{1}{f_s} = \tau_s + \tau_{si} \tag{6}$$

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.

$$\tau_f \approx \tau_{F \to S} + (N_s - 1) \cdot T_s + \tau_s + C_f \tag{7}$$

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 6 above.

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}.

$$T_f = \frac{1}{f_f} = \tau_f + \tau_{fi} \tag{8}$$

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. 8 is met.

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 7 below.

Table 7: Approximate fixed overhead times.

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

2.9 In-depth topics

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

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 10 below shows this operation.

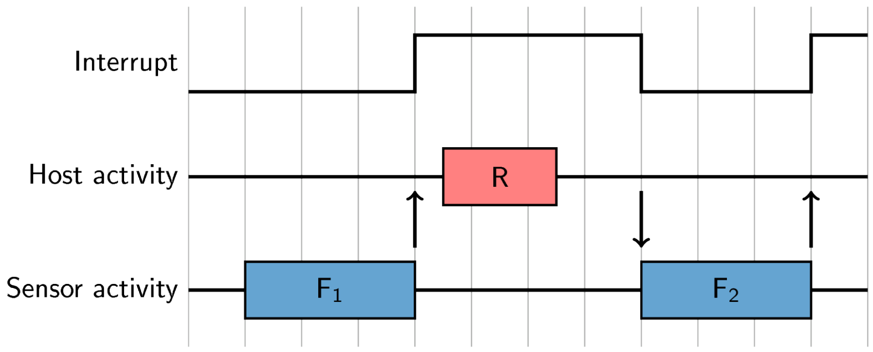

Figure 10: Illustration of basic control flow of the A121. The top part shows the interrupt pin. The middle part shows what the host is doing, where a red box denoted with an R is a *read*. The bottom part shows what the sensor is doing, where a blue box denoted with a F is a frame measurement. A down arrow shows a *measure* call, and an up arrow shows a completed measurement.

Note that in the case shown in Figure 10, 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 11 shows this method of operation.

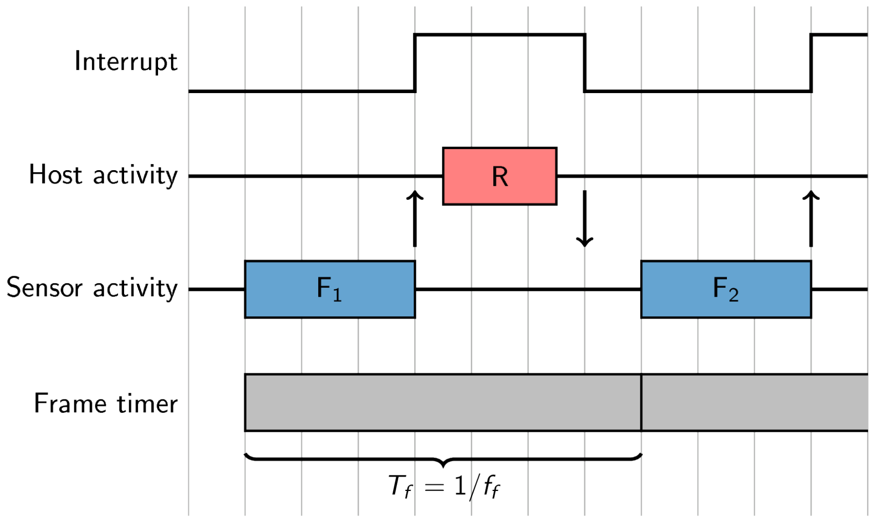

Figure 11: Illustration of a control flow of the A121 where the *frame rate* is set. For context, see Figure 10. The bottom part shows the frame timer set up according to the frame rate.

In the case shown in Figure 11, 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 12 below.

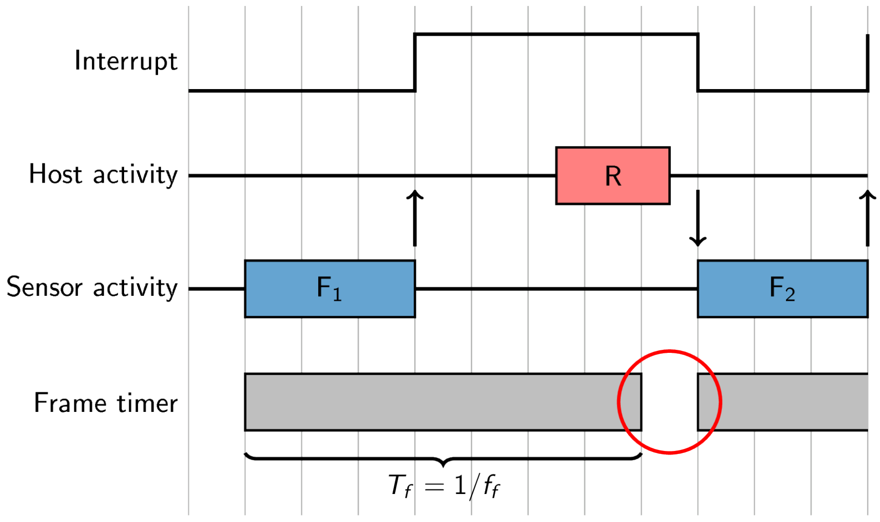

Figure 12: Illustration of a control flow of the A121 where the *frame rate* is set and the host responds late. This makes the frame *delayed*. For context, see Figure 11.

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.

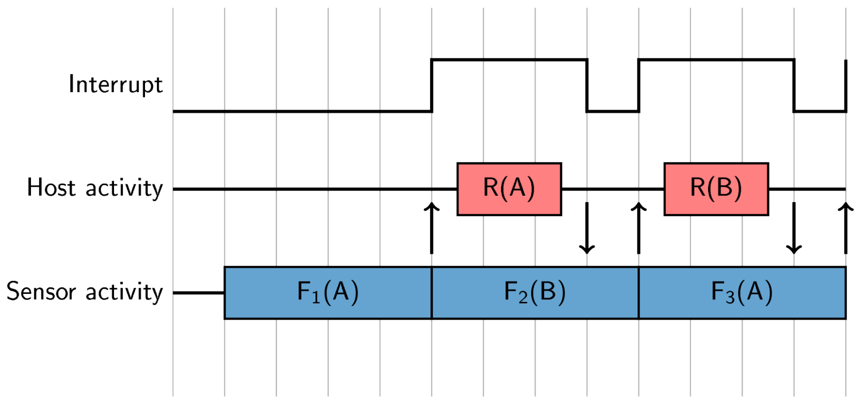

Figure 13: Illustration of a control flow of the A121 using *double buffering*.

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.

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`.

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.

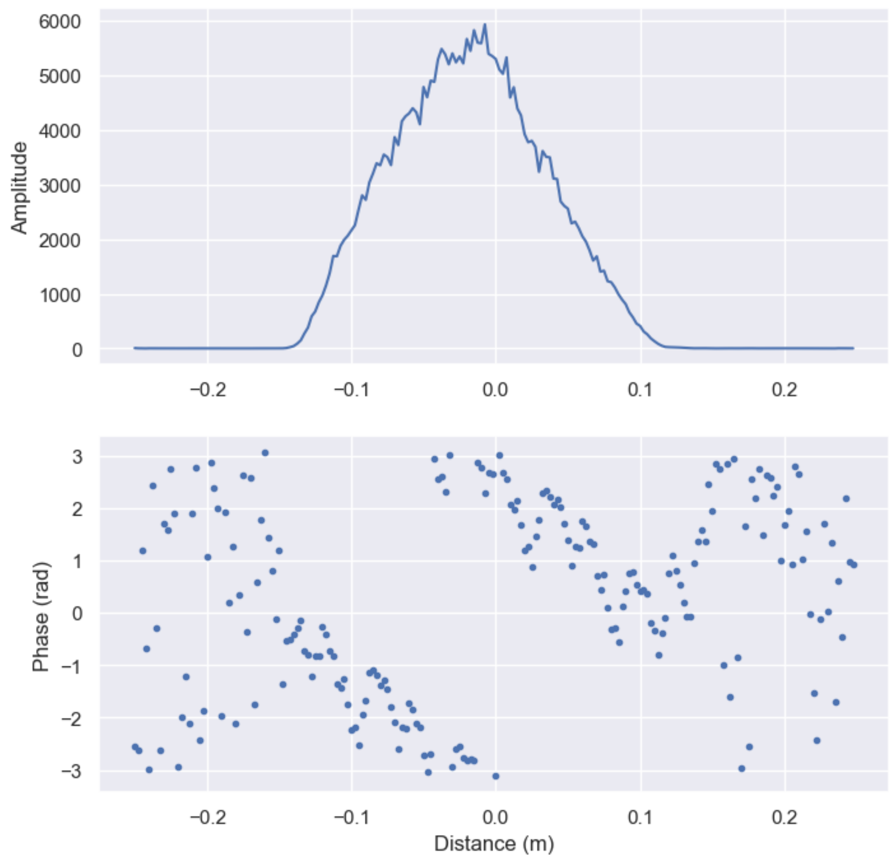

Figure 14: Upper graph: Amplitude of loopback measurement. Lower graph: Phase of loopback measurement.

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.

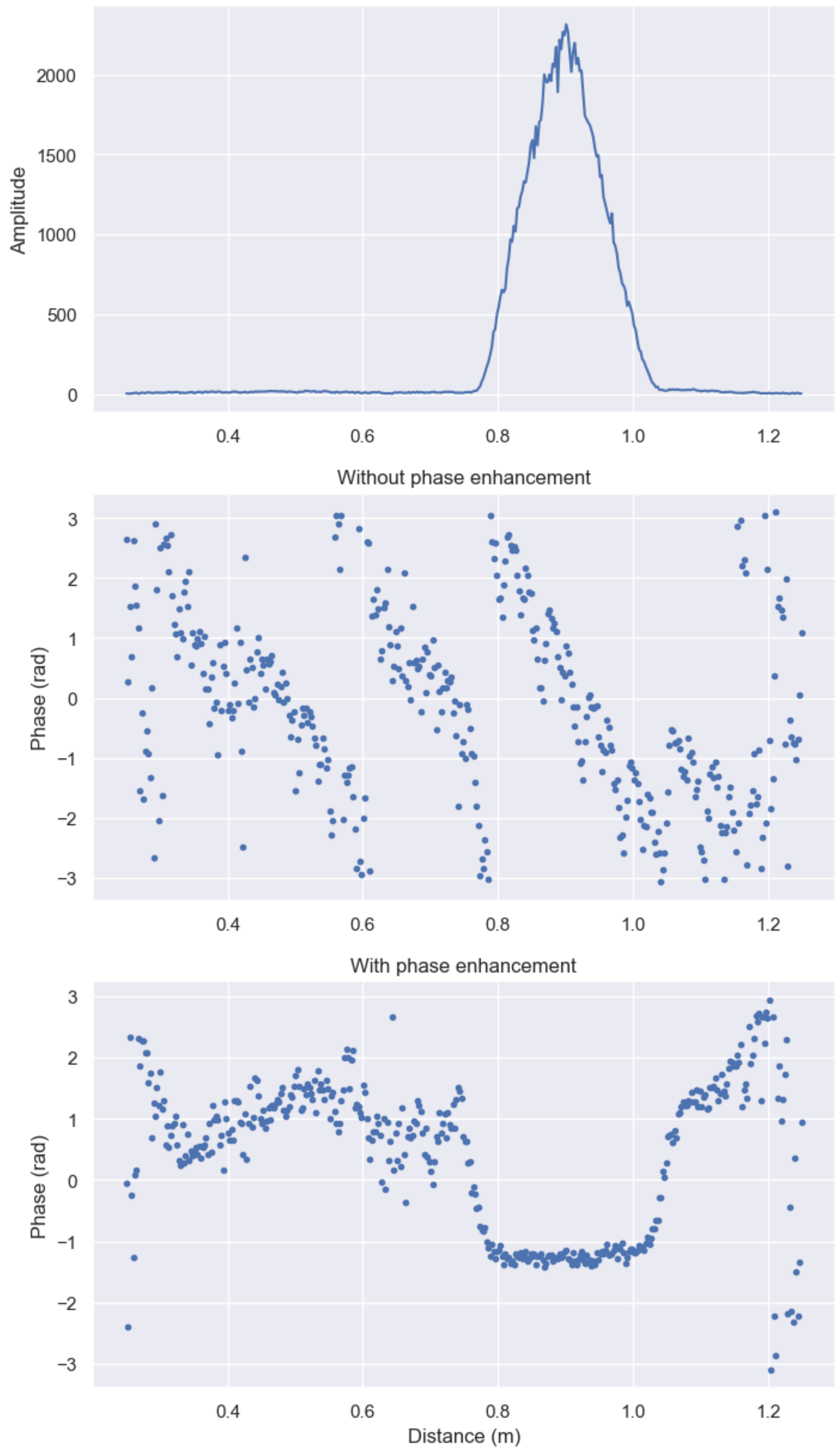

Figure 15: Upper graph: Amplitude of measured sweep. Middle graph: Phase of measured sweep with phase enhancement disabled. Lower graph: Phase of measured sweep with phase enhancement enabled.

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.

IQ imbalance compensation

Key features **#2** and **#4** are used to achieve consistent envelope amplitude values in the distance domain.

The phase rotates at a distance point when an object moves within a small interval and corresponds to where in the interval the object is located. IQ imbalance causes oscillation in amplitude when the phase rotates. Amplitude oscillation will cause the center of gravity of a peak to shift and impair distance estimation. Compensating for the IQ imbalance will reduce the amplitude oscillation.

Loopback measurements are performed to quantify the IQ imbalance at all distance points in a small interval. The result is used to calculate compensation coefficients, one for each distance point in the interval. The complex samples from a distance point are adjusted by multiplication with the corresponding coefficients. The IQ-imbalance repeats outside the small interval so the same coefficients are reused for all distances.

Figures below show the phase rotation and resulting amplitude, with and without IQ imbalance compensation, for an object that moves within the interval. The left and right envelope corresponds to the same object but with different phase rotations.

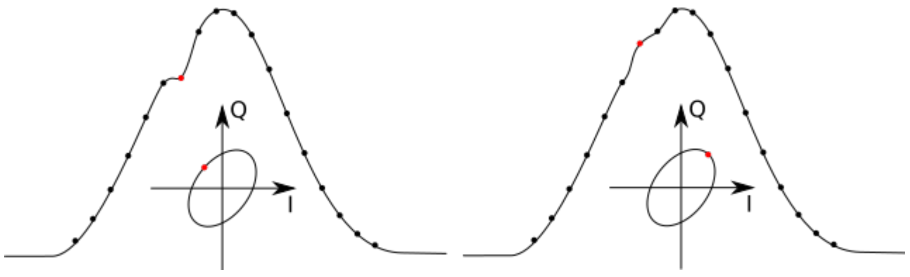

Figure 16: Amplitude oscillates when the phase rotates over a distance without compensation.

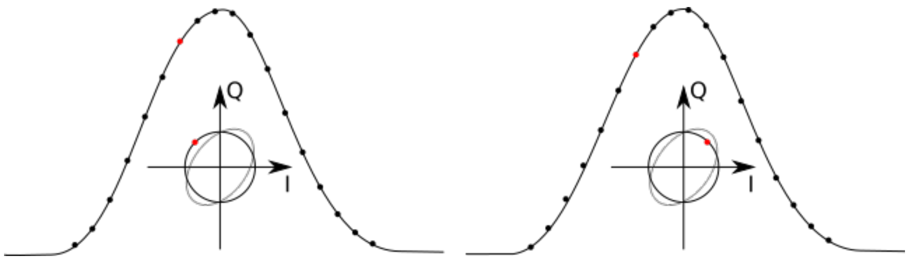

Figure 17: Amplitude does not oscillate when the phase rotates over a distance with compensation.

The IQ imbalance compensation 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.

Part I

Appendix

3 Disclaimer