A121 Presence Detector

User Guide

A121 Presence Detector

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 Presence Detector

This presence detector measures changes in the data over time to detect motion. It is divided into two separate parts:

Intra-frame presence – detecting (faster) movements *inside* frames
For every frame and depth, the intra-frame deviation is based on the deviation from the mean of the sweeps

Inter-frame presence – detecting (slower) movements *between* frames
For every frame and depth, the absolute value of the mean sweep is filtered through a fast and a slow low pass filter. The inter-frame deviation is the deviation between the two filters and this is the base of the inter-frame presence. As an additional processing step, it is possible to make the detector even more sensitive to very slow motions, such as breathing. This utilizes the phase information by calculating the phase shift in the mean sweep over time. By weighting the phase shift with the mean amplitude value, the detection of slow moving objects will increase.

Both the inter- and the intra-frame deviations are filtered in time. Also, to be more robust against changing environments and variations between sensors, normalization is done against the noise floor. Finally, the output from each part is the maximum value in the measured range.

Presence detected is defined as either inter- or intra-frame detector having a presence score above chosen thresholds.

2.1 How to use

Tuning the sensor parameters

A large part of the presence detector consists of automatic configuration of the sensor parameters. This can of course be overridden, but it is recommended to use the automatic configuration for best performance.

Detection Range

The most important parameter that the user needs to adjust is the range: *start_m* and *end_m*. The start parameter has a major effect on the automatic configuration, it is therefore important to adjust the start point to be as far from the sensor as possible, while still fulfilling the requirements for the use case. Avoid adding range close to the sensor without justification, since this will have negative impact on both power consumption and performance. The *end_m* parameter should also not be further away from the sensor than the use case requires. A common pitfall is to have an unnecessarily long range, which can have unexpected effects, for example detections from static objects and walls in the background. When a person moves around, a wall might suddenly “appear” after being blocked by the person. This will have the effect that the wall then appears to be moving and be detected by the presence detector.

Automatic Subsweep Selection

If the *automatic_subsweeps* is set to True, the sensor will automatically be configured with several subsweeps with different `hwaas` and possibly different `profile` for each subsweep. This is the recommended way to configure the detector, since it minimizes power consumption as well as smoothing out detection levels over distances.

When using the automatic subsweep selection, we still need to set the *signal_quality* parameter. The higher signal quality, the higher power consumption. It is recommended to set the value so that the highest HWAAS is different for the furthest subsweeps, i.e. if both subsweep 3 and 4 have maximized HWAAS to 511, this means that the signal quality is better for subsweep 3 than for subsweep 4.

Configuring the sensor manually

If the automatic subsweep selection is not activated, a single subsweep will instead be used. This means that the same *profile* and *hwaas* will be used for the whole range. The limiting factor will be the *start_m*, which determines which profile can be used. The profile is set to the biggest profile with no direct leakage in the chosen range. This is to maximize SNR. The shortest start range needed for the different profiles can be found in Table 2:

Table 2: Minimum start range for different profiles.

Profile	Start range
1	0 m
2	0.14 m
3	0.28 m
4	0.38 m
5	0.64 m

Note

To maximize SNR in long range detections, the start range needs to be set to at least 0.64 m.

For each profile a half power pulse width can be calculated based on the pulse length. We choose the *step_length* to not exceed this value, while still having it as long as possible. We want the step length as long as possible to reduce power consumption, but short enough to get good SNR in the whole range. Choosing a high number of *hwaas* will increase SNR. However, it will also affect the power consumption. Choose the highest possible HWAAS that still fulfills your power requirements. A good starting point is to use the default value. For better use of the intra-frame presence detector, increase the number of *sweeps_per_frame*. This will improve the sensitivity.

Tuning the detector parameters

To adjust overall sensitivity, the easiest way is to change the thresholds. There are separate thresholds for the inter-frame and the intra-frame parts, *inter_detection_threshold* and *intra_detection_threshold*. If only one of the motion types is of interest, the intra-frame and inter-frame presence can be run separately, otherwise they can be run together. The detection types are enabled with the *inter_enable* and *intra_enable* parameters.

If a stable detection and fast loss of detection is important, for example when a person is leaving the sensor coverage, the *inter_frame_presence_timeout* functionality can be enabled. If the inter-frame presence score has declined during a complete timeout period, the score is scaled down to get below the threshold faster.

Advanced detector parameters

Another way to adjust overall sensitivity is to change the output time constants. Increase time constants to get a more stable output or decrease for faster response.

Fast motions - looking for a person walking towards or away from the sensor

The intra-frame part has two parameters: *intra_frame_time_const* and *intra_output_time_const*.

Look at the depthwise presence plot in the GUI. If it can't keep up with the movements, try decreasing the intra frame time constant. Instead, if it flickers too much, try increasing the time constant. Furthermore, if the presence score output flickers too much, try increasing the intra output time constant, while on the other hand decreasing it will give faster detection.

Slow motions - looking for a person resting on a sofa

For the base functionality, the inter-frame part has four parameters: *inter_frame_slow_cutoff*, *inter_frame_fast_cutoff*, *inter_frame_deviation_time_const*, and *inter_output_time_const*.

The inter-frame slow cutoff frequency determines the lower frequency cutoff in the filtering. If it is set too low, unnecessary noise might be included, which gives a higher noise floor, thus decreasing sensitivity. On the other hand, if it is set too high, some very slow motions might not be detected.

The inter-frame fast cutoff frequency determines the higher bound of the frequency filtering. If it is set too low, some faster motions might not be detected. However, if it is set too high, unnecessary noise might be included. Values larger than half the *frame_rate* disables this filter. If that is not enough, you need a higher frame rate or to use the intra-frame part.

Inter-frame timeout

For faster loss of detection, *inter_frame_presence_timeout* can be used. This regulates the number of seconds needed with decreasing inter-frame presence score before the score starts to get scaled down faster. If set to low, the score might drop when a person sits still and breathes slowly. If set very high, it will have no effect.

2.2 Detailed description

The sparse IQ service service returns data frames in the form of N_s sweeps, each consisting of N_d range distance points, see Frames, sweeps and subsweeps. We denote frames captured using the sparse IQ service as $x(f,s,d)$, where f denotes the frame index, s the sweep index and d the range distance index.

Intra-frame detection basis

For very fast motions and fast detection we have the intra-frame presence detection. The idea is simple – for every frame we depth-wise take the deviation from the sweep mean and low pass (smoothing) filter it.

Let N_s denote the number of sweeps, and let the deviation from the mean be:

$$s_{\text{intra\_dev}}(f,d) = \sqrt{\frac{N_s}{N_s - 1}} \cdot \frac{1}{N_s} \sum_s |x(f,s,d) - y(f,d)|$$

where the first factor is a correction for the limited number of samples (sweeps).

Then, let the low pass filtered (smoothed) version be:

$$\bar{s}_{\text{intra\_dev}}(f,d) = \alpha_{\text{intra\_dev}} \cdot \bar{s}_{\text{intra\_dev}}(f-1,d) + (1 - \alpha_{\text{intra\_dev}}) \cdot s_{\text{intra\_dev}}(f,d)$$

The smoothing factor α_{intra} is set through the *intra_frame_time_const* parameter.

The relationship between time constant and smoothing factor is described under *Calculating smoothing factors*.

The intra-frame deviation is normalized with a noise estimate.

Inter-frame detection basis

In the typical case, the time between *frames* is far greater than the time between *sweeps*. Typically, the frame rate is 2 - 100 Hz while the sweep rate is 3 - 30 kHz. Therefore, when looking for slow movements in presence, the sweeps in a frame can be regarded as being sampled at the same point in time. This allows us to take the mean value over all sweeps in a frame, without losing any information. In the basic part of the inter frame presence, we only use the amplitude value. Let the *absolute mean sweep* be denoted as

$$y(f,d) = |\frac{1}{N_s} \sum_s x(f,s,d)|$$

We take the mean sweep y and depth-wise run it through two *exponential smoothing* filters (first order IIR low pass filters). One slower filter with a larger smoothing factor, and one faster filter with a smaller smoothing factor. Let α_{fast} and α_{slow} be the smoothing factors and $\bar{y}_{\text{fast}}$ and $\bar{y}_{\text{slow}}$ be the filtered sweep means. For every depth d in every new frame f:

$$\bar{y}_{\text{slow}}(f,d) = \alpha_{\text{slow}} \cdot \bar{y}_{\text{slow}}(f-1,d) + (1 - \alpha_{\text{slow}}) \cdot y(f,d)$$
$$\bar{y}_{\text{fast}}(f,d) = \alpha_{\text{fast}} \cdot \bar{y}_{\text{fast}}(f-1,d) + (1 - \alpha_{\text{fast}}) \cdot y(f,d)$$

The relationship between cutoff frequency and smoothing factor is described under *Calculating smoothing factors*.

From the fast and slow filtered absolute sweep means, a deviation metric $s_{\text{inter\_dev}}$ is obtained by taking the absolute deviation between the two:

$$s_{\text{inter\_dev}}(f,d) = \sqrt{N_s} \cdot |\bar{y}_{\text{fast}}(f,d) - \bar{y}_{\text{slow}}(f,d)|$$

Where $\sqrt{N_s}$ is a normalization constant. In other words, $s_{\text{inter\_dev}}$ relates to the instantaneous power of a band-pass filtered version of y. This metric is then filtered again with a smoothing factor, $\alpha_{\text{inter\_dev}}$, set through the *inter_frame_deviation_time_const* parameter, to get a more stable metric:

$$\bar{s}_{\text{inter\_dev}}(f,d) = \alpha_{\text{inter\_dev}} \cdot \bar{s}_{\text{inter\_dev}}(f-1,d) + (1 - \alpha_{\text{inter\_dev}}) \cdot s_{\text{inter\_dev}}(f,d)$$

This is the basis of the inter-frame presence detection. As with the intra-frame deviation, it's favorable to normalize this with the noise floor.

Noise estimation

To normalize detection levels, we need an estimate of the noise power generated by the sensor. We assume that from a static channel, i.e., a radar signal with no moving reflections, the noise is white and its power is its variance. However, we do not want to rely on having such a measurement to obtain this estimate.

Since we're looking for motions generated by humans and other living things, we know that we typically won't see fast moving objects in the data. In other words, we may assume that *high frequency content in the data originates from sensor noise*. Since we have a relatively high sweep rate, we may take advantage of this to measure high frequency content.

Extracting the high frequency content from the data can be done in numerous ways. The simplest to implement is possibly a FFT, but it is computationally expensive. Instead, we use another technique which is both robust and cheap.

First, to remove any trends from fast motion in the frame, we differentiate over the sweeps $N_{\text{diff}} = 3$ times:

$$x'(f,s,d) = x^{(1)}(f,s,d) = x(f,s,d) - x(f,s-1,d)$$

$$\dots$$

$$x^{(N_{\text{diff}})}(f,s,d) = x^{(N_{\text{diff}}-1)}(f,s,d) - x^{(N_{\text{diff}}-1)}(f,s-1,d)$$

Then, take the mean absolute deviation:

$$\hat{n}(f,d) = \frac{1}{N_s - N_{\text{diff}}} \sum_{s=1+N_{\text{diff}}}^{N_s} |x^{(N_{\text{diff}})}(f,s,d)|$$

And normalize such that the expectation value would be the same as if no differentiation was applied:

$$n(f,d) = \hat{n}(f,d) \cdot \left[\sum_{k=0}^{N_{\text{diff}}} \binom{N_{\text{diff}}}{k}^2 \right]^{-1/2}$$

Finally, apply an exponential smoothing filter with a smoothing factor α_{noise} to get a more stable metric:

$$\bar{n}(f,d) = \alpha_{\text{noise}} \cdot \bar{n}(f-1,d) + (1 - \alpha_{\text{noise}}) \cdot n(f,d)$$

This smoothing factor is set from a fixed time constant of 10 s.

Both the intra-frame deviation, $\bar{s}_{\text{intra\_dev}}(f,d)$, and the inter-frame deviation, $\bar{s}_{\text{inter\_dev}}(f,d)$ are normalized by the noise estimate, $\bar{n}(f,d)$, as:

$$\bar{s}(f,d) = \frac{\bar{s}(f,d)}{\bar{n}(f,d)}$$

Output and distance estimation

The outputs from the noise normalized intra-frame deviation and inter-frame deviation are the maximum scores of the respective deviation:

$$v(f) = \max_d(z(f,d))$$

As a final step, the outputs are low pass filtered:

$$\bar{v}(f) = \alpha_{\text{output}} \cdot \bar{v}(f-1) + (1 - \alpha_{\text{output}}) \cdot v(f)$$

The smoothing factors for the outputs are set through the `intra_output_time_const` and the `inter_output_time_const` parameters.

When both detectors are enabled, presence is defined as either the intra-frame or the inter-frame being over the threshold. If both have detection, the faster nature of intra-frame presence compared to inter-frame presence makes it best practice to use this score to estimate distance. If only one part has detection we will use this for the distance estimate. The estimate is based on the peak value in the data. Let p be the “present”/”not present” output and d_p be the presence depth index output:

$$p = v > v_{\text{threshold}}$$

$$d_p = \arg\max_d(z(f,d))$$

Inter-frame timeout

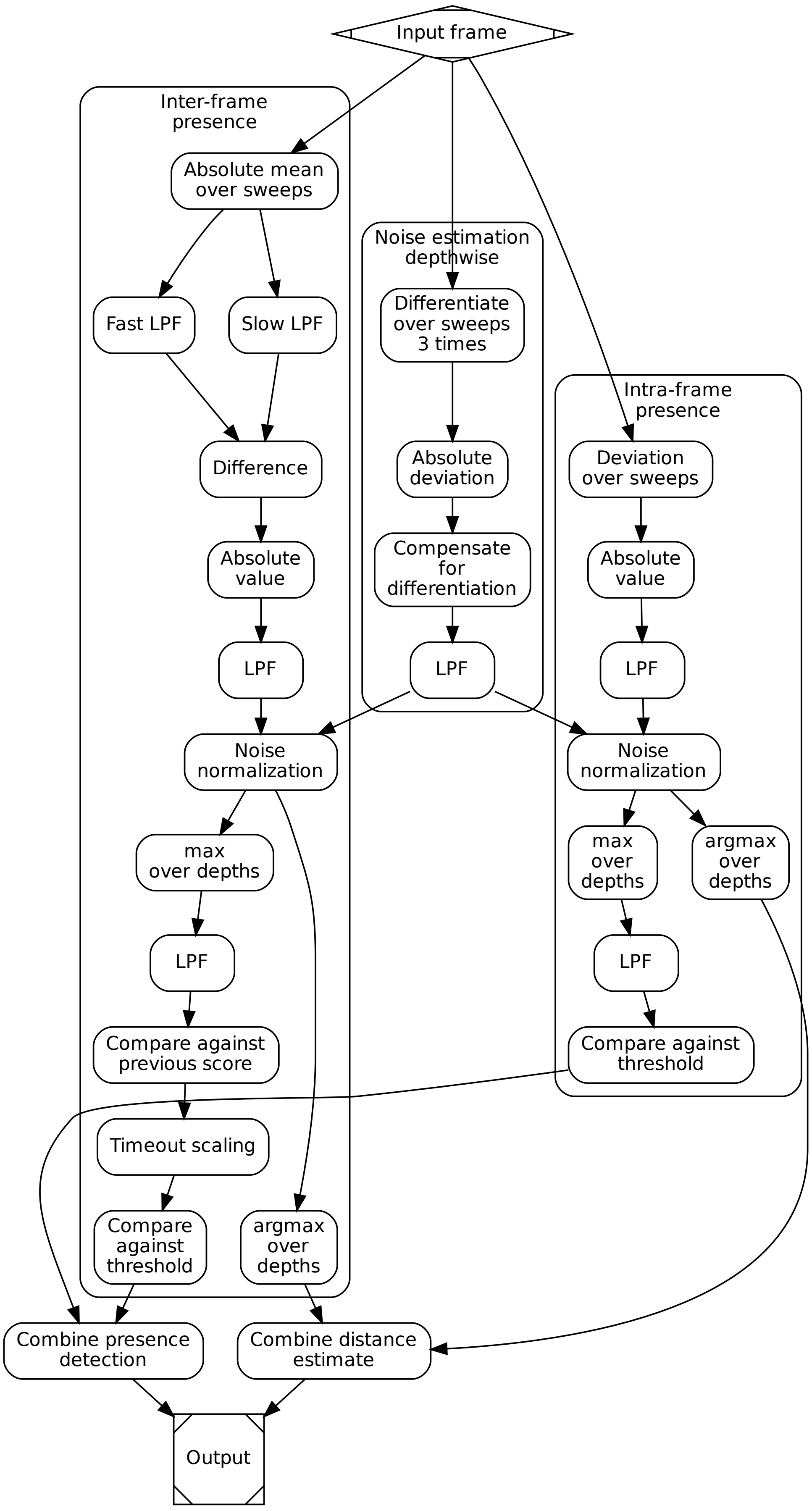
For faster decline of the inter-frame presence score, an exponential scaling of the score starts after t seconds determined by the `inter_frame_presence_timeout` parameter. We track the number of frames with declining score, n. With the frame rate defined as f_f, the scale factor, C_{inter}, is calculated as:

$$C_{\text{inter}} = \exp\left(\frac{\max(n - (t \cdot f_f), 0)}{t \cdot f_f}\right)$$

And the inter-frame presence score is scaled as:

$$\bar{v}_{\text{inter}}(f) = \frac{\bar{v}_{\text{inter}}(f)}{C_{\text{inter}}}$$

Graphical overview



Calculating smoothing factors

Instead of directly setting the smoothing factor of the smoothing filters in the detector, we use cutoff frequencies and time constants. This allows the configuration to be independent of the frame rate.

The symbols used are:

Symbol	Description	Unit
α	Smoothing factor	1
τ	Time constant	s
f_c	Cutoff frequency	Hz
f_f	Frame rate	Hz

Going from time constant τ to smoothing factor α:

$$\alpha = \exp\left(-\frac{1}{\tau \cdot f_f}\right)$$

The bigger the time constant, the slower the filter.

Going from cutoff frequency f_c to smoothing factor α:

$$\alpha = \begin{cases} 2 - \cos(2\pi f_c/f_f) - \sqrt{\cos^2(2\pi f_c/f_f) - 4\cos(2\pi f_c/f_f) + 3} & \text{if } f_c < f_f/2 \\ 0 & \text{otherwise} \end{cases}$$

The lower the cutoff frequency, the slower the filter. The expression is obtained from setting the -3 dB frequency of the resulting exponential filter to be the cutoff frequency. For low cutoff frequencies, the more well known expression $\alpha = \exp(-2\pi f_c/f_f)$ is a good approximation.

Read more: time constants, cutoff frequencies.

2.3 Hints and Recommendations

This section contains some practical considerations for how to configure the presence detector optimally.

Range settings

Start by estimating the range settings for your use-case. A common pitfall is to let the range be too extensive, which can lead to the detector triggering from movement in unexpected locations. In a similar manner, setting the range too close to the sensor can cause the automatic configuration to dedicate unnecessary resources to search in ranges where there won't be any movement. So aim to let the range cover the range where the movement is expected to occur, but not beyond that.

When the range settings have been selected, it is recommended to use the subsweep selection to set the appropriate values for HWAAS and profile.

An interesting phenomenon that occurs when the range is longer than necessary is indirect detections from movement. If an object blocking the sensor is removed, this might cause an object further away (like a wall) to suddenly appear after being blocked. This will be interpreted as movement, since the object moved into view.

Adjusting Threshold

The threshold is very dependent on the use case, the most natural way to adjust this is by testing relevant scenarios. A too low threshold will cause false positives from unwanted movement. Setting the threshold too high will cause missed detections instead. A good starting point is to estimate roughly what the noise level is for your use case. This is done by measuring an empty channel and observing the highest presence score during the measurement, any threshold below this value will be completely useless, since it will constantly trigger false detections.

Smoothing filter and latency

When the threshold and range settings are deemed satisfactory, the smoothing filters can be addressed. The smoothing filters have a direct impact on the latency of the detector. The trade-off is between latency and retention, a long filter will take more time to detect a movement, but retain detection and avoid "flickering" behavior. A short filter will drop detection more frequently, but also gain detection faster. This is a general behavioral aspect of the detector, which should be adjusted according to the use case. For some applications, it might be relevant to have retention built into an application on top of the detector instead of using the built in filters.

3 C API

The focus of this section is the Presence Detector C API.

It is recommended to read this section together with example_detector_presence.c located in the SDK package. The full API specification, rss_api.html, provided in the SDK package is also good to read.

The Presence Detector will utilize a single sensor configuration with multiple sweeps in every frame to detect motion.

3.1 Configuration

The Presence Detector is controlled using configuration parameters. All parameters will be shown in Table 3 but some will be described in more detail in this section.

automatic_subsweeps Automatic subsweps will divide the measurement range in different subsweeps to optimize `profile`, `step_length` and `hwaas`. `signal_quality` is only used when `automatic_subsweeps` is enabled and it will affect `hwaas` to increase or decrease the signal quality. `automatic_subsweeps` will disable the following settings in the detector:

- `auto_profile_enabled`
- `auto_step_length_enabled`
- `manual_profile`
- `manual_step_length`
- `hwaas`

It will also invalidate the following parts in the metadata, since they will be different for different subsweeps:

- `step_length_m`
- `profile`

auto_profile_enabled By default, the best fit for the profile is calculated from the start of the range, `start_m`. This can be overridden by setting `auto_profile_enabled` to `false` and setting `manual_profile`.

auto_step_length_enabled By default, the best fit for the `step_length` is calculated from the profile. This can be overridden by setting `auto_step_length_enabled` to `false` and setting `manual_step_length`.

frame_rate_app_driven By default, the `frame_rate` is maintained by the sensor. In the low power use case when one wants to disable the sensor between measurements, the application will have to make sure the measurements are performed at the rate set by `frame_rate`.

Presets

The Presence Detector in example_detector_presence.c is configured through presets. A preset is a set of configuration parameters tuned for a certain use case. The presets used in this example are *Medium Range*, *Short Range*, *Long Range*, and *Low Power Wakeup*. Default preset is *Medium Range*.

Configuration Parameters

Name	Type	Default Value	Min	Max
inter_frame_presence_timeout	uint16_t	3		
intra_detection_enabled	bool	true	n/a	n/a
inter_detection_enabled	bool	true	n/a	n/a
intra_detection_threshold	float	1.3	0.0	5.0
inter_detection_threshold	float	1.0	0.0	5.0
inter_frame_deviation_time_const	float	0.5	0.01	20.0
inter_frame_fast_cutoff	float	6.0	1.0	50.0
inter_frame_slow_cutoff	float	0.2	0.01	1.0
intra_frame_time_const	float	0.15	0.0	1.0
intra_output_time_const	float	0.3	0.01	20.0
inter_output_time_const	float	2.0	0.01	20.0
sensor_id	sensor id	1	n/a	n/a
auto_profile_enabled	bool	true	n/a	n/a

auto_step_length_enabled	bool	true	n/a	n/a
manual_profile	enum	profile_1	profile_1	profile_5
manual_step_length	uint16_t	24		
start_m	float	0.3		< end_m
end_m	float	2.5	> start_m	
sweeps_per_frame	uint16_t	16		
hwaas	uint16_t	32	1	511
inter_frame_idle_state	enum	idle_state_deep_sleep	idle_state_deep_sleep	idle_state_ready
frame_rate	float	12.0		
frame_rate_app_driven	bool	false	n/a	n/a
reset_filters_on_prepare	bool	true	n/a	n/a
automatic_subsweeps	bool	true	n/a	n/a
signal_quality	float	20.0	-10.0	60.0

Table 3: Presence Detector Configuration Parameters

3.2 Detector Result

The result from a call to acc_detector_presence_process() includes both the presence result as well as the complete Sparse IQ Service result. This section will only describe the presence result.

result member	type	description
presence_detected	bool	true if presence was detected, false otherwise
intra_presence_score	float	A measure of the amount of fast motion detected
intra_presence_score	float	A measure of the amount of slow motion detected
presence_distance	float	The distance, in meters, to the detected object
depthwise_intra_presence_scores	float array	An array of measures of the amount of fast motion detected per distance point.
depthwise_inter_presence_scores	float array	An array of measures of the amount of slow motion detected per distance point.
depthwise_presence_scores_length	uint32_t	The number of elements in the depthwise presence scores arrays
processing_result	struct	Described in Sparse IQ Service User Guide

3.3 Memory

Flash

The example application compiled from example_detector_presence.c on the XM125 module requires around 80 kB.

RAM

The RAM can be divided into three categories, static RAM, heap, and stack. Below is a table for approximate RAM for an application compiled from example_detector_presence.c for different presets.

RAM	**Size (kB)**			
Preset	*Medium*	*Short*	*Long*	*Wakeup*
Static	1	1	1	1
Heap	6	6	6	4
Stack	4	4	4	4
Total	11	11	11	9

Note that the heap is very dependent on the preset. The configurations that have the largest impact on the memory are start_m, end_m, step_length and sweeps_per_frame.

3.4 Power Consumption

The example application compiled from example_detector_presence_low_power_hibernate.c on the XM125 module has an average current of 5.3 mA.

The example application compiled from example_detector_presence_low_power_hibernate.c with preset *Low Power Wakeup* has an average current of 0.07 mA on the XM125 module.

4 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.