A121 Presence Detector

User Guide

A121 Presence Detector

User Guide

Author: Acconeer AB

Version:a121-v1.3.0

Acconeer AB October 6, 2023

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 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
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
	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

2 Presence detection

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 both in time and depth. 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

First, the range of detection needs to be determined. Based on the start range, *start_m*, a best fit for the profile is calculated. 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 deault 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.

For slow motion detection, there is the possibility to use *inter_phase_boost* to increase sensitivity. This will increase detection for someone sitting still and breathing, even if the sensor is not placed in an optimal position. However, have in mind that it will increase detection of all slow moving objects.

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

Antoher 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 phase boost To increase detection of very slow motions `inter_phase_boost` can be enabled.

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 Handbook. 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 depthwise 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 (smoothened) 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 and, when appropriate, a depth filter is applied, both are discussed in later sections.

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 depthwise run it though 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 bandpass 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-fram deviation, it's favorable to normalize this with the noise floor and, if relevant, apply a depth filter. Both are discussed in later sections.

Inter-frame phase boost

To increase detection of very slow motions, we utlize the phase information in the Sparse IQ data. The first step is to calculate the phase shift over time. Let $u(f,d)$ be the *mean sweep*:

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

The mean sweep is low pass filtered and the smoothing factor, $\alpha_{\text{for\_phase}}$, is set from a fixed and quite high time constant, τ_{for_phase}, of 5 s:

$$\bar{u}_{\text{for\_phase}}(f,d) = \alpha_{\text{for\_phase}} \cdot \bar{u}_{\text{for\_phase}}(f-1,d) + (1-\alpha_{\text{for\_phase}}) \cdot u(f,d)$$

When a new frame is sampled, we take the mean sweep and calculate the phase shift between this mean sweep and the previous low pass filtered mean sweep. We define the phase shift to never exceed π radians by adding $2\pi k$ for some integer k:

$$\phi(f,d) = |angle(u(f,d)) - angle(\bar{u}_{\text{for\_phase}}(f,d)) + 2\pi k|$$

In open air where only noise is measured, the phase will jump around. To amplify the phase shift boost for human breathing, while at the same time decreasing it for open air, the phase shift is weighted with the amplitude. For a more stable weighting, the mean sweep is low pass filtered before the amplitude is calculated:

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

$$A(f,d) = |\bar{u}_{\text{for\_amp}}(f,d)|$$

The amplitude is noise normalized(see next section) and truncated to reduce unwanted detections from very strong static objects:

$$A(f,d) = \max(A(f,d), 15)$$

Before the final output is generated, the depthwise inter-frame presence score is multiplied with the phase and amplitude weight:

$$\bar{s}_{\text{inter\_dev}}(f,d) = \bar{s}_{\text{inter\_dev}}(f,d) \cdot \phi(f,d) \cdot A(f,d)$$

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, $\bar{s}_{\text{inter\_dev}}(f,d)$, as well as the amplitude in the ínter-frame phase boost is normalized by the noise estimate, $\bar{n}(f,d)$, as:

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

Depth filtering

If we choose profile and step length in a way that the reflection spans several depth points, we apply a depth filter with length n on both the noise normalized intra-frame deviation, and the noise normalized inter-frame deviation. If the depth filter length is odd we have:

$$n' = \frac{n-1}{2}$$

$$z(f,d) = \frac{1}{2n'+1} \sum_{i=-n'}^{n'} \bar{s}(f,d+i)$$

and if the depth filter length is even we have:

$$n' = \frac{n}{2}$$

$$z(f,d) = \frac{1}{2n'} \sum_{i=-n'}^{n'-1} \bar{s}(f,d+i)$$

where the signal $\bar{s}$ is zero-padded, i.e.:

$$\bar{s}(f,d) = 0 \text{ for } d < 1 \text{ or } d > N_d$$

Output and distance estimation

The outputs from the noise normalized and depth filtered 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 smooting 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-fram presence makes it best practise 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 fram 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:

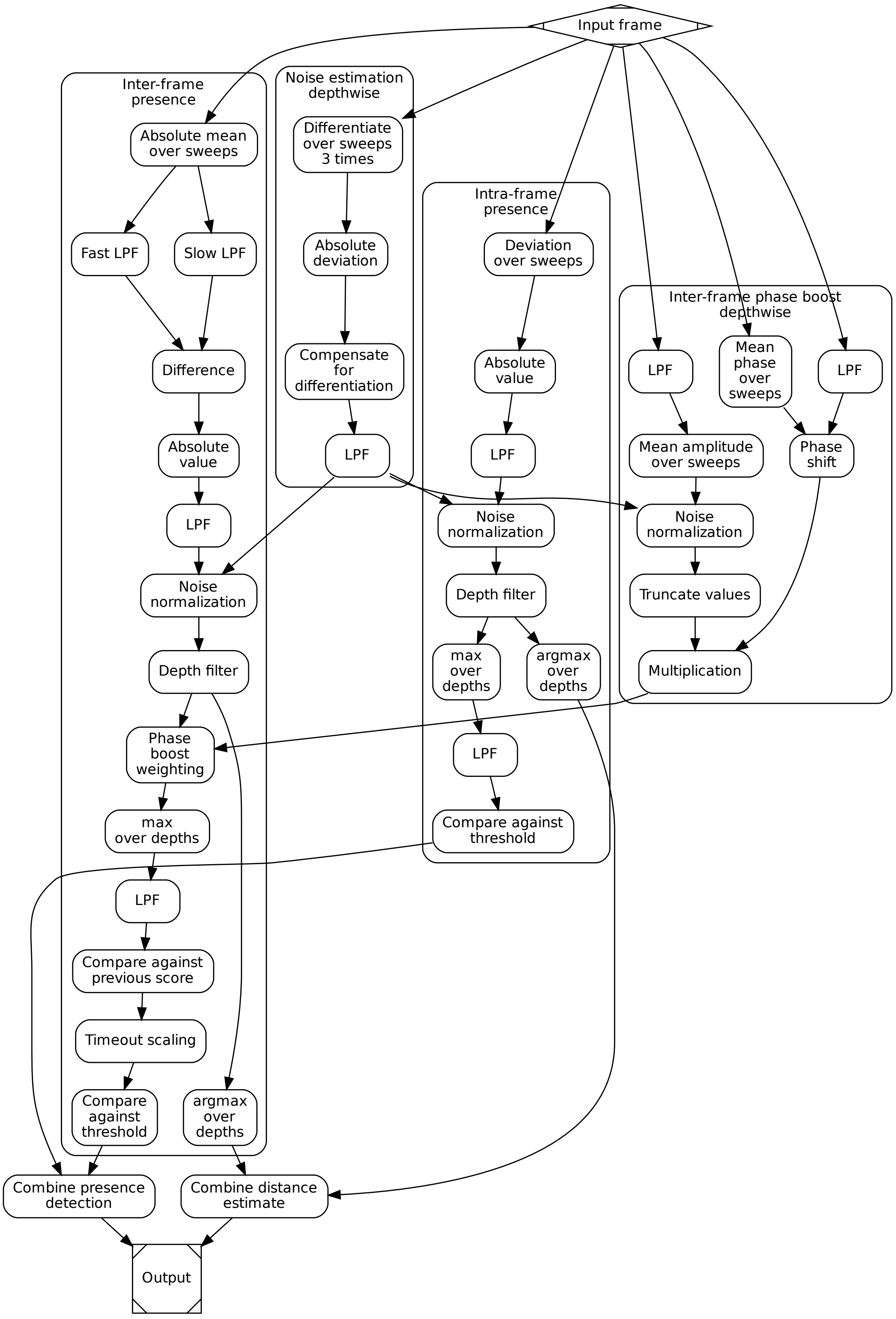
$$\bar{v}_{\text{inter}}(f) = \frac{\bar{v}_{\text{inter}}(f)}{C_{\text{inter}}}$$

To reduce the effect of the inter-frame phase boost when the score is scaled, the time constant, $\tau_{\text{for\_phase}}$, controlling the smoothing factor $\alpha_{\text{for\_phase}}$, is scaled in a similar way. With scale factor C_τ, the time constant, τ_{scaled}, is calculated as:

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

$$\tau_{\text{scaled}} = \frac{\tau_{\text{for\_phase}}}{C_\tau}$$

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.

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 3.1 but some will be described in more detail in this section.

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

Table 3: Presence Detector Configuration Parameters

Name	Type	Default Value	Min	Max
sweeps_per_frame	uint16_t	16		
inter_frame_presence_timeout	uint16_t	3		
inter_phase_boost_enabled	bool	false	n/a	n/a
intra_detection_enabled	bool	true	n/a	n/a
inter_detection_enabled	bool	true	n/a	n/a
frame_rate	float	12.0		
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_4	profile_1	profile_5
manual_step_length	uint16_t	72		
start_m	float	0.3		< end_m
end_m	float	2.5	> start_m	
frame_rate_app_driven	bool	false	n/a	n/a
reset_filters_on_prepare	bool	true	n/a	n/a
inter_frame_idle_state	enum	deep_sleep	deep_sleep	ready
hwaas	uint16_t	32	1	511

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	7.0	6	6
Stack	3	3	3	3
Total	10	11	10	10

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 11.4 mA.

The example application compiled from example_detector_presence_low_power_hibernate.c with preset *Low Power Wakeup* has an average current of 0.13 mA on the XM125 module.

4 Disclaimer