A121 Distance Detector

User Guide

A121 Distance Detector

User Guide

Author: Acconeer AB

Version:a121-v1.5.0

Acconeer AB January 19, 2024

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 STM32CubeIDE	Describes the flow of taking an Acconeer SDK and integrate into STM32CubeIDE.	- Using STM32CubeIDE
A121 Raspberry Pi Software	Describes how to develop for Raspberry Pi.	- Working with Raspberry Pi
A121 Ripple	Describes how to develop for Ripple.	- Working with Ripple on Raspberry Pi
XM125 Software	Describes how to develop for XM125.	- Working with XM125
XM126 Software	Describes how to develop for XM126.	- Working with XM126
I2C Distance Detector	Describes the functionality of the I2C Distance Detector Application.	- Working with the I2C Distance Detector Application
I2C Presence Detector	Describes the functionality of the I2C Presence Detector Application.	- Working with the I2C Presence Detector Application
I2C Breathing Reference Application	Describes the functionality of the I2C Breathing Reference Application.	- Working with the I2C Breathing Reference Application
	Handbook (PDF)	
Handbook	Describes different aspects of the Acconeer offer, for example radar principles and how to configure	- To understand the Acconeer sensor - Use case evaluation
	Readme (txt)	
README	Various target specific information and links	- After SDK download

2 Distance detection

The goal of the distance detector is to produce highly accurate distance measurements while maintaining low power consumption by combining the features of the A121 sensor with powerful signal processing concepts, all wrapped with a simple to use API.

The full functionality can be explored in the Exploration Tool. Once the desired performance is achieved, the configuration can be carried over to the embedded version of the algorithm, available in the C-SDK.

2.1 Introduction

The purpose of the distance detector is to detect objects and estimate their distance from the sensor. The algorithm is built on top of the Sparse IQ service and has various configuration parameters available to tailor the detector to specific use cases. The detector utilizes the following key concepts:

1. Distance filtering: A matched filter is applied along the distance dimension to improve the signal quality and suppress noise.

2. Subsweeps: The measured range is split into multiple subsweeps, each configured to maintain SNR throughout the sweep while minimizing power consumption.

3. Comparing sweep to a threshold: Peaks in the filtered sweep are identified by comparison to one of three available threshold methods.

4. Estimate distance to object: Estimate the distance to the target by interpolation of the peak and neighboring amplitudes.

5. Sort found peaks: If multiple peaks are found in a sweep, three different sorting methods can be employed, each suitable for different use-cases.

2.2 Distance filter

As the sensor produce coherent data, samples corresponding to the location of an object will have similar phase, while the phase of free-air measurements will be random. By applying a filter in the distance domain, the noise in the free-air regions will be suppressed, resulting in an improved SNR.

The filter is automatically configured based on the detector configuration as a second order Butterworth filter with a cutoff frequency corresponding to a matched filter.

2.3 Subsweeps

The measurement range is split up into multiple subsweeps to allow for optimization of power consumption and signal quality. The profile, HWAAS and step length are automatically assigned per subsweep, based on the detector config.

- A shorter profile is selected at the start of the measurement range to minimize the interference with direct leakage, followed by longer profiles to gain SNR. The longest profile used can be limited by setting the parameter `max_profile`. If no profile is specified, the subsweeps will be configured to transfer to the longest profile(without interference from direct leakage) as quickly as possible to maximize SNR. Longer profiles yield a higher SNR at a given power consumption level, while shorter profiles gives better depth resolution.
- The step length can also be limited by setting the parameter `max_step_length`. If no value is supplied, the step length is automatically configured to appropriate size, maintaining good depth resolution while minimizing power consumption. Note, the algorithm interpolates between the measured points to maintain good resolution, even with a more coarse step length.
- HWAAS is assigned to each subsweep in order to maintain SNR throughout the measured range as the signal strength decrease with the distance between the sensor and the measured target. The target SNR level is adjusted using the parameter `signal_quality`.

  Note, higher signal quality will increase power consumption and measurement time.

  The expected reflector shape is considered when assigning HWAAS to the subsweeps. For planar reflectors, such as fluid surfaces, select `PLANAR`. For all other reflectors, select `GENERIC`.

In the Exploration Tool GUI, the subsweeps can be seen as slightly overlapping lines. If the measured object is in the overlapping region, the result from the neighboring segments is averaged together.

2.4 Thresholds

To determine if any objects are present, the sweep is compared to a threshold. A peak is defined as a middle point that has greater amplitude than its two neighbouring points. For an object to be detected, it has to yield a peak where all three points are above the threshold. Three different thresholds can be employed, each suitable for different use-cases.

Fixed amplitude threshold The simplest approach to setting the threshold is choosing a fixed threshold over the full range. The amplitude value is set through the parameter *fixed_threshold_value*. The fixed amplitude threshold does not have any temperature compensation built in.

Fixed strength threshold This threshold takes a fixed strength value and converts to the corresponding amplitude value. The purpose is to produce a threshold that is able to detect an object of a with a specific reflectiveness, independent of the distance to the object. The strength value is set through the parameter *fixed_strength_threshold_value*. The fixed strength threshold does not have any temperature compensation built in.

Recorded threshold In situations where stationary objects are present, the background signal is not flat. To isolate objects of interest, the threshold is based on measurements of the static environment. The first step is to collect multiple sweeps, from which the mean sweep and standard deviation is calculated. Secondly, the threshold is formed by adding a number of standard deviations (the number is determined by the parameter *threshold_sensitivity*) to the mean sweep. The recorded threshold has a built in temperature compensation, based on the internal temperature sensor.

Constant False Alarm Rate (CFAR) threshold (default) A final method to construct a threshold for a certain distance is to use the signal from neighbouring distances of the same sweep. This requires that the object gives rise to a single strong peak, such as a fluid surface and not, for example, the level in a large waste container. The main advantage is that the memory consumption is minimal. The sensitivity of the threshold is controlled through *threshold_sensitivity*. As the CFAR threshold is formed based on each momentary sweep, any temperature effects on the signal are implicitly accounted for by the algorithm.

2.5 Reflector shape

The expected reflector shape is considered when assigning HWAAS to the subsweeps and during peak sorting.

The reflector shape is set through the detector configuration parameter *reflector_shape*.

For a planar reflector, such as a fluid surface, select *PLANAR*. For all other reflectors, select *GENERIC*.

2.6 Reflector strength

The reflector strength characterize the reflectiveness of the detected object. The detector reports a strength number for each estimated distance.

The strength is estimated using the RLG equation, peak amplitude, noise floor estimate and the sensor base RLG. More information on the RLG equation and base RLG can be found here.

The estimated strength is used by the detector when sorting the estimated distances according to their relative strengths. It can also be used by the application to infer information about a certain distance estimate. For example, a highly reflective object such as a metal surface will typically have a higher strength number than a less reflective surface such as a wooden structure.

Ideally, the strength estimate is agnostic to the distance of the object. However, due to close range effects, the strength tends to be under estimated at short distances (< 1m).

The strength is reported in dB.

2.7 Peak sorting

Multiple objects in the scene will give rise to several peaks. Peak sorting allows selection of which peak is of highest importance.

The peak sorting strategy is set through *PeakSortingMethod*, which is part of the detector configuration.

The following peak sorting options are available.

Closest This method sorts the peaks according to distance from the sensor.

Strongest (default) This method sorts the peaks according to their relative strength.

Note, the reflector shape is considered when calculating each peak's strength. The reflector shape is selected through detector configuration parameter *reflector_shape*.

Note, regardless of selected peak sorting strategy, all peaks and the corresponding strenghts are returned by the distance detector.

2.8 Detector calibration

For optimal performance, the detector performs a number of calibration steps. The following section outlines the purpose and process of each step. Note, which of the following calibration procedures to perform is determined by the user provided detector config. For instance, the close range measurement is only performed when measuring close to the sensor.

To trigger the calibration process in the Exploration Tool gui, simply press the button labeled "Calibrate detector". If you are running the detector from a script, the calibration is performed by calling the method *calibrate_detector*.

Noise level estimation The noise level is estimated by disabling of the transmitting antenna and just sample the background noise with the receiving antenna. The estimate is used by the algorithm for various purposes when forming thresholds and estimating strengths.

Offset compensation The purpose of the offset compensation is to improve the distance trueness(average error) of the distance detector. The compensation utilize the loopback measurement, where the pulse is measured electronically on the chip, without transmitting it into the air. The location of the peak amplitude is correlated with the distance error and used to correct the distance raw estimate.

Close range measurement calibration Measuring the distance to objects close to the sensor is challenging due to the presence of strong direct leakage. One way to get around this is to characterize the leakage component and then subtract it from each measurement to isolate the signal component. This is exactly what the close range calibration does. While performing the calibration, it is important that the sensor is installed in its intended geometry and that there is no object in front of the sensor as this would interfer with the direct leakage.

Note, this calibration is only performed if close range measurement is active, given by the configured starting point.

Recorded threshold The recorded threshold is also recorded as a part of the detector calibration. Note, this calibration is only performed if the detector is configured to used recorded threshold or if close range measurement is active, where recorded threshold is used.

2.9 Detector recalibration

To maintain optimal performance, the sensor should be recalibrated if *sensor_calibration_needed* is set to True. A sensor calibration should be followed by a detector recalibration, performed by calling *recalibrate_detector*.

The detector recalibration carries out a subset of the calibration steps. All the calibration steps performed are agnostic to its surroundings and can be done at any time without considerations to the environment.

2.10 Temperature compensation (Recorded threshold)

The surrounding temperature impacts the amplitude of the measured signal and noise. To compensate for these effects, the recorded threshold has a built in compensation model, based on a temperature measurement, internal to the sensor. Note, the effectiveness of the compensation is limited when measuring in the close range region.

The CFAR threshold exhibits an indirect temperature compensation as the threshold is formed based on the sweep itself. As the sweep changes with temperature, so does the threshold accordingly.

The fixed thresholds(amplitude and strength) does not have any temperature compensation.

2.11 Result

The result return by the distance detector is contained in the class `DetectorResult`.

The two main components of the distance detector result are the estimated `distances` and their corresponding estimated reflective `strengths`. The distances and the corresponding strengths are sorted according to the selected peak sorting strategy.

In addition to the distances and strengths, the result also contains the boolean `near_edge_status`. It indicates if an object is located close to start of the measurement range, but not resulting in a clear peak, but rather the tail of an envelope. The purpose of the boolean is to provide information in the case when an object is present, just outside of the measurement range. One example of when this becomes useful is the Tank reference application, which is built on top of the distance detector. If the tank is overflowing, the peak might end up just outside of the measured interval, but the tail end of the envelope would still be observable.

The result also contains the boolean `sensor_calibration_needed`. If True, the procedure, described in the section Detector Recalibration, needs to be performed to maintain optimal performance.

Note, the sweep and threshold, presented in the distance detector GUI are not returned by the distance detector. These entities are processed and evaluated internally to the algorithm. The purpose of visualizing them in the GUI is to guide in the process of determining the detector configuration, such as selection of threshold strategy and sensitivity.

3 C API

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

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

The Distance Detector utilizes one or more sensor configurations to cover the full configured range. This will result in multiple sensor measurements for one detector result. Thereby, multiple detector functions are called in a while loop waiting for a sensor interrupt for each iteration.

An example of how to use the API is provided in the SDK: example_detector_distance.c

3.1 Calibration

There are two types of calibrations needed to use the distance detector.

- Sensor calibration
- Detector calibration

The sensor calibration ensures that the sensor can measure properly. The detector calibration ensures that the calculated distances are correct.

The sensor calibration should be performed before the detector calibration.

Environment and Temperature Constraints

Since the distance detector needs to be calibrated, there are some constraints when using it in different physical environments and temperatures.

Sensor Calibration

The sensor calibration is not dependent on the physical environment. As an example, a sensor calibration can be done with the sensor placed in one part of a tank but is still valid if the sensor is moved to another part of the tank.

The sensor calibration is dependent on temperature. If the temperature changes more than 15 degrees Celsius, the sensor calibration needs to be redone.

See the Sparse IQ User Guide for more information on the sensor calibration.

Detector Calibration

The detector calibration is dependent on the physical environment if at least one of the following configurations is used:

- Close range leakage cancellation: true (default false)
- Threshold method: Recorded (default CFAR)

This means that if at least one of these two configurations is used, the detector calibration must be done in the setup where it will be used. Objects present in front of the sensor during the detector calibration in this configuration will not be detected during normal operation.

Whenever a sensor calibration needs to be redone (due to a temperature change), a detector calibration update needs to be done. The detector calibration update is a subset of a full calibration and is not dependent on the physical environment. This means that, for example, objects within the measurement range during the calibration update will still be detected after the calibration update.

The close range leakage cancellation part of the calibration is not included in the calibration update, even though it is temperature dependent. The reason for this is that it is also dependent on the physical environment, as mentioned above. This means that close range leakage cancellation cannot be used in applications where the temperature changes more than 15 degrees Celsius during operation.

Summary

Below is a table of the dependency to the physical environment and/or temperature depending on configuration and which calibration function is used (full or update).

Configuration	**Full Calibration**	**Calibration Update**
Close range leakage cancellation	- Physical environment - Temperature	- Invalid
Recorded threshold	- Physical environment - Temperature	- Temperature
Other	- Temperature	- Temperature

No Retention

Note that if there's no retention in the application, the full calibration needs to be done every time before measuring. This means that close range leakage cancellation and/or recorded threshold cannot be used unless caching of the calibration result is done. See section Calibration Caching for more information.

RSS API Usage

The calibration function handles all sensor communication within the detector, except for waiting for sensor interrupt. The calibration is performed in multiple steps and therefore the function needs to be called in a while loop until complete.

See example_detector_distance.c for how to do this.

Calibration Caching

When reading this section, it is good to look at example_detector_distance_calibration_caching.c located in the SDK package.

Calibration caching is typically done to reduce power consumption in applications where temperature changes are common or if there's no memory retention in the application. Calibration caching means that the results are saved and then used later instead of redoing the calibration.

There are three types of calibration results.

- Sensor calibration result
- Static detector calibration result
- Dynamic detector calibration result

The static detector calibration result is temperature independent. This means that it only needs to be saved once and can then be used regardless of temperature.

The sensor calibration result and the dynamic detector calibration result are temperature dependent. This means that they need to be saved for a specific temperature. If the sensor needs to be re-calibrated, the saved calibration results can be used instead of doing a new calibration. The saved calibration results should only be used if they were produced within a temperature range of at most +- 15 degrees Celsius from the current temperature.

The implementation of calibration caching needs to be done in the application, i.e. it is not part of the RSS library itself. To see an example of how it can be done, please look in example_detector_distance_calibration_caching.c.

3.2 Process

Depending on the configuration the Distance Detector will use one or more sensor configurations resulting in one or more sensor measurements for each detector measurement. The process function also requires a specific call chain to be performed for one sensor measurement. This call chain should be performed within a while loop to cover all possible sensor measurements.

Sparse IQ Data

As part of the distance result struct there is a member called processing_result which contains the underlying Sparse IQ data used to calculate the distance result. The processing_result will be updated each time the acc_detector_distance_process function is called.

3.3 Memory

Flash

The example application compiled from example_detector_distance.c on the XM125 module requires around 90 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_distance.c.

RAM	Size (kB)
Static	1.0
Heap	15.0
Stack	3.3
Total	19.3

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

3.4 Power Consumption

The example application compiled from example_detector_distance_low_power_off.c on the XM125 module has an average current of 0.27 mA.

4 Configuration Parameters

Table 4: Distance Detector Configuration Parameters

Name	Type	Default Value	Min	Max
sensor	sensor id	1	n/a	n/a
start_m	float	0.25	0.0	< end_m
end_m	float	3.0	> start_m	23.0
max_step_length	uint16_t	0		
max_profile	enum	profile_5	profile_1	profile_5
signal_quality	float	15.0	-10.0	35.0
threshold_method	enum	cfar		
peak_sorting_method	enum	strongest		
reflector_shape	enum	generic		
num_frames_in_recorded_threshold	uint16_t	100		
fixed_amplitude_threshold_value	float	100.0		
fixed_strength_threshold_value	float	0.0		
threshold_sensitivity	float	0.5	0.0	1.0
close_range_leakage_cancellation	bool	false	n/a	n/a

5 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.