A121 Parking Reference Application User Guide

User Guide

A121 Parking Reference Application User Guide

User Guide

Author: Acconeer AB

Version:a121-v1.7.0

Acconeer AB June 4, 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 Parking Reference Application	Describes the functionality of the Parking Reference Application.	- Working with the Parking Reference Application
A121 STM32CubeIDE	Describes the flow of taking an Acconeer SDK and integrate into STM32CubeIDE.	- Using STM32CubeIDE
A121 Raspberry Pi Software	Describes how to develop for Raspberry Pi.	- Working with Raspberry Pi
A121 Ripple	Describes how to develop for Ripple.	- Working with Ripple on Raspberry Pi
XM125 Software	Describes how to develop for XM125.	- Working with XM125
XM126 Software	Describes how to develop for XM126.	- Working with XM126
I2C Distance Detector	Describes the functionality of the I2C Distance Detector Application.	- Working with the I2C Distance Detector Application
I2C Presence Detector	Describes the functionality of the I2C Presence Detector Application.	- Working with the I2C Presence Detector Application
I2C Breathing Reference Application	Describes the functionality of the I2C Breathing Reference Application.	- Working with the I2C Breathing Reference Application
	Handbook (PDF)	
Handbook	Describes different aspects of the Acconeer offer, for example radar principles and how to configure	- To understand the Acconeer sensor - Use case evaluation
	Readme (txt)	
README	Various target specific information and links	- After SDK download

2 Parking

The use case of mounting a radar on a parking space is an established functional concept present in the Acconeer offering for the A111 sensor. This algorithm extends this to the A121 sensor as well.

2.1 Algorithm Concept

This algorithm has an intended use of determining whether a stationary car is present in front of the sensor as well as maintaining a low power consumption. The main idea is that a large stationary object will reflect a similar amount of energy at the same distance over time, whereas a moving object (like a human) will have a more varied reflection. To achieve this detection, we introduce a measurement, called “signature” of the sweep. The signature is calculated so that a small change in amplitude or depth of the reflection will create a small change in the signature. So two signatures that are close to each other will also correspond to similar reflected energy and thus likely the same object. The algorithm proceeds to collect signatures over time and determine if a certain number of signatures are close to each other.

The measurement used for this algorithm is similar in nature to the measurement of the distance detector, which also detects (and reports distance to) static objects. However, this configuration is tailor made for the parking use case and designed with a power constraint in mind.

The signatures are calculated from the mean sweep over a frame. For all sweeps, we only use the amplitude measurements for each depth, denoted with $A(d)$ (which is obtained by taking the absolute value of the sparseIQ data). The phase information is not used in this reference application. See handbook-a121-spf for more information about frames and sweeps.

2.2 Integration Notes

This use case is highly dependent on the integration, both for the obstruction detection and the primary parking detection. All notes here should be taken as general guidelines.

In the pole mounted case (when the sensor looks at the car from the side) there are a few issues found in testing. Since a car hood often consists of large flat surfaces which can deflect the signal instead of reflecting it, there is a risk that a car can appear “invisible” to the sensor. The most natural way to mitigate this issue is to tilt the sensor slightly downwards, which often catches the front of the car, which more often than the hood has reflective surfaces more perpendicular to the sensor. All pole mounted tests were performed with the sensor angled slightly downwards, see Figure 2 and Figure 1.

Figure 1: Illustration of the sensor mount with angle used in testing the pole mounted case.

For the ground mounted case, the sensor should be looking straight up and be placed as central (under the car) as possible. A common issue is that the casing creates some reflections within the closer parts of the detection range, which in turn can cause false detects. This can be mitigated by either adjusting the start of the range (recommended) or adjusting the *amplitude_threshold*. Adjusting the amplitude_threshold can impact detection performance.

Another caveat is to set the range too far from the sensor, so that the start of the range is inside the car. This can cause the object to be missed, since the signal will reflect on the bottom of the car and travel too far in the allotted time. So the start of the range should be kept as low as possible while not encountering problems with the integration. The best way to achieve this is by testing using Exploration Tool.

Signature

The signature concept mentioned here and used in the algorithm is a 2D measure on the amplitude data of the whole sweep. It differs slightly between the obstruction detection and the normal parking detector by taking the average amplitude over the whole sweep in the obstruction case while taking the max amplitude in the parking detection case. Here WD denotes "Weighted Distance".

$$Avg(A) = \frac{1}{N_d}\sum_d A(d)$$

$$Max(A) = \max_d(A(d))$$

$$WD(A) = \frac{\sum_d A(d) * d}{\sum_d A(d)}$$

So for the parking case, the signature S_p, is obtained by:

$$S_p(A) = WD(A), Max(A)$$

And for the obstruction detection, the signature S_o:

$$S_o(A) = Avg(A), Max(A)$$

For the parking case, the measurement is not necessarily continuous. But in practice, small changes in amplitude tends to result in small changes in the signature, which is the relevant property used for the algorithm.

2.3 Obstruction Detection

In addition to the parking detection, the algorithm also has a component called “Obstruction detection”. This system measures a few points very close to the sensor, well within the so called “direct leakage”. An obstruction of the sensor will result in a change of the received direct leakage and/or returned signal, causing the signature of the data to change. The obstruction detection allows the user to configure the sensitivity as well as the range of the processing. Since the amplitude is sensitive to temperature fluctuations, it is recommended to test the threshold in appropriate conditions for the use case. If the obstruction detection feature is used, it is important that the range of the obstruction detection does not overlap with the range of the parked car detection.

2.4 Measurement Range and Presets

The lower bound for the range is limited to 0.15 m, which ensures that the measurement is outside the direct leakage, which can interfere with detection. The upper limit is set to 5 m due to usage of the standard prf (15.6 MHz).

We have two main presets, one for where the sensor is located close to the car (typically ground or curb mounted) and looking up at the car. And one for when the sensor is looking at the car at a longer range (typically the case of a pole mounted sensor) and looking at the car from the side or slightly downwards. These two presets are aptly named “Ground” and “Pole” mounted. There is also a preset called “Ground_Low_Energy” which is intended to reflect a case with high focus on power requirements optimization.

Note that these settings are intended to reflect a “best practice”, the settings can of course be manually adjusted to more extreme ranges with the caveat that the functionality might suffer.

2.5 Configuration

The most basic configuration parameters for the reference application is the range settings (*range_start_m* and *range_end_m*). These determine the approximate range from the sensor wherein we expect to find a part of a car. Note that it is detrimental to the performace to let the range be to close to the sensor, it is not recommended to set the *range_start_m* closer than the direct leakage allows, which is constricted in Exploration tool as well. If an object is blocking the sensor, even if the range setting is beyond it, it will still affect the signal.

Ref App Parameters

This section contains all parameters that directly pertain to the reference application functionality.

The *queue_length_n* parameter sets how many consecutive detections above the threshold that is needed to report a parked car. Note that this behaviour is closely related to the update rate, with a high update rate, it is natural to increase the number of required detections for parking detection.

The *amplitude_threshold* determines how many times the noise level the amplitude level needs to be in order to consider the signature as detected.

Variable *weighted_distance_threshold_m* determines the maximum weighted distance difference between two signatures to retain detection.

The *signature_similarity_threshold* determines a percentage of signatures that needs to be within the *weighted_distance_threshold_m* distance to retain detection. If set to 100%, we require all samples to be within range in order to retain detection. This value allows for example a person to pass the detector (which briefly changes the signature) without dropping detection.

Obstruction Detection Parameters

The intended usage of the obstruction detection is to determine if an object is blocking the sensor. This is achieved by sampling a few samples to establish a signature of the signal in a very close range then comparing this signature to future signatures on the same range. The signature might drift over time due to temperature changes, which means that we can either model this drift or recalibrate with regular intervals. The parameters listed in this section concern this functionality. If the *obstruction_detection* is false, the functionality is not active and can be ignored.

The *obstruction_distance_threshold* variable determines how large fraction of each dimension the obstruction measurement can vary without considering the sensor obstructed. E.g. if the *obstruction_distance_threshold* is set to 0.04 and the range is between 0.02 m and 0.05 m. The weighted distance part of the signature can vary (0.05 m - 0.02 m) * 0.04 = 0.0012 m.

The two last variables determine the range where we can detect obstruction: *obstruction_start_m* and *obstruction_end_m*. The detection range should not intersect with the detection range for the main functionality.

Sensor Settings

There are three settings which correspond directly to the SparseIQ counterparts, for detailed description see: handbook-a121-spf. The most significant is the *update_rate*, for a battery driven application, this is the main factor behind the energy consumption.

The *hwaas* parameter has an impact on the overall signal to noise ratio (SNR), low values will yield a noisy signal and poorer performance, since the signature will vary more due to noise variations.

The *profile* determines the length of the pulse. If the range is within the so called “direct leakage”, the performance of the algorithm will deteriorate. For close ranges (i.e. ground mounted applications), it is recommended to use profile 1 or 2. For longer ranges, higher profiles should be used. The limitations that the range puts on profile choices is implemented in the Exploration tool GUI and can be experimented with there.

2.6 Calibration

Estimation of the underlying noise level is an important factor for the performance of the algorithm. This is done at start up with an empty channel. The purpose of this is to get a noise estimate in the current temperature. If the obstruction detection is activated, we also get the signature for the unobstructed close range. The processing has an internal model to account for temperature changes during execution. However, for large temperature fluctuations, there is still a risk that the obstruction detection performance deteriorates.

Note that the calibration of the obstruction detection must be done with the sensor unobstructed in the intended casing, since the calibration takes a snapshot of the radar signal in the obstruction detection range.

In general, it is better to calibrate the sensor as close as possible to operating temperatures.

Temperature

The SNR is dependent on the sensor temperature, this relationship is modelled in the algorithm and compensated for. The general rule is that the SNR increases in lower temperatures and decreases with increased temperatures. However, this is based on the temperature during calibration. So for optimal performance, the calibration temperature should be as close to operating conditions as possible.

2.7 Results

The reference application will provide one or two main results: detection and if activated, obstruction. In addition, the ref app provides detailed information that is mainly used for plotting, but can of course be used for other purposes within your application.

The *car_detected* indicates whether the algorithm detects a parked car within the range.

The *obstruction_detected* indicates if the algorithm detects an obstruction of the sensor. If the obstruction detection is not activated in the configuration, this will always be False.

The *extra_result* contains information mainly used for plotting. This result is not available in the embedded implementation for a microcontroller.

2.8 Testing

Three different cases have been tested: Ground, Pole and Curb mounted.

The pole mounted case intends to simulate when the sensor is mounted inside an electric charging station or similar where the sensor is looking at the car from the front (or back), either at an angle or direct ahead. A preset for the pole mounted case can also be found among the presets in Exploration tool. This case was tested with an FZP lens, see Figure 2.

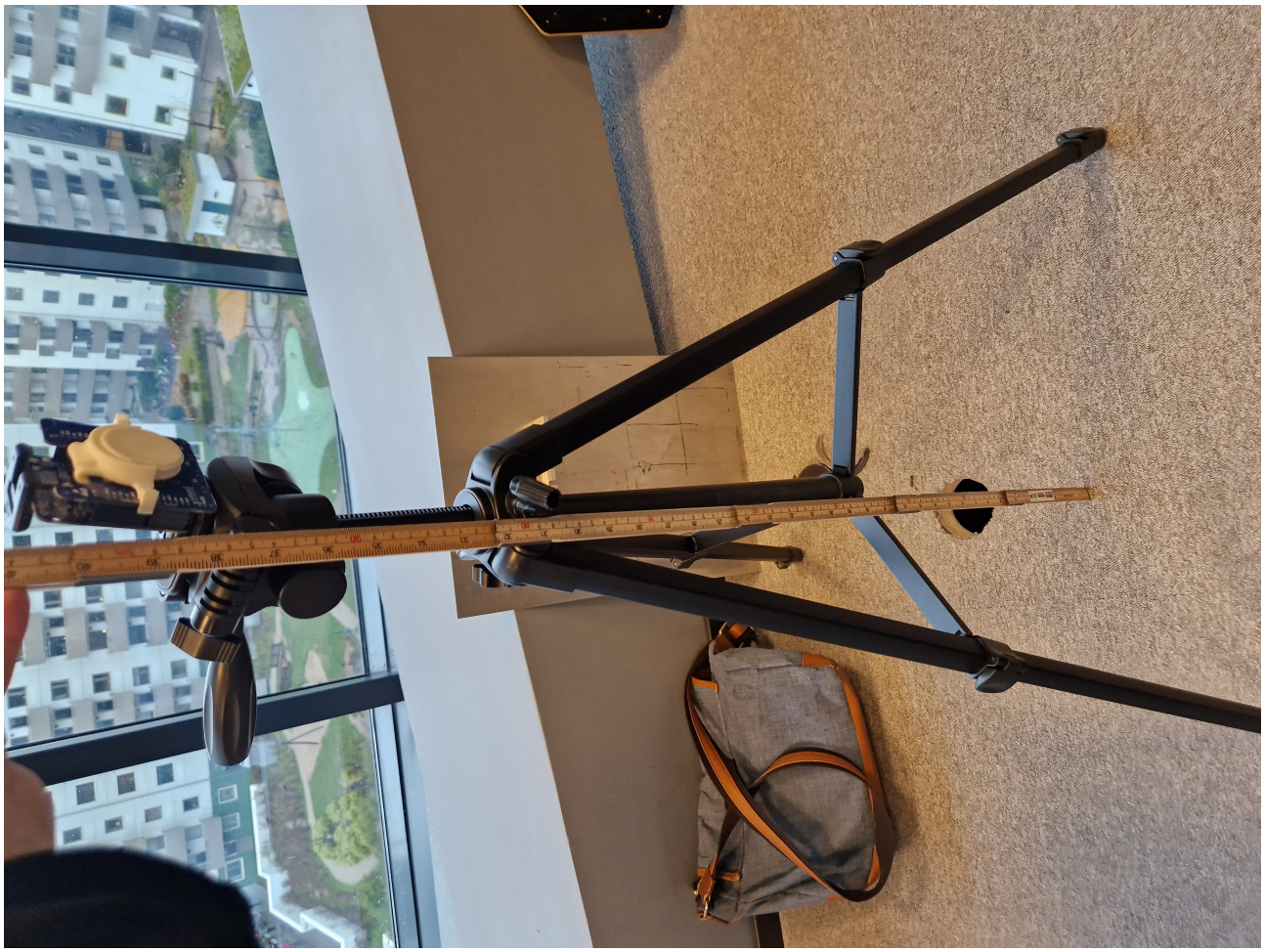

Figure 2: Test setup for the pole mounted testing, note that the sensor is tilted slightly downwards (about 20 degrees from a ground perpendicular axis). An FZP lens was also used, as seen in the picture.

The ground and curb mounted case is tested when the sensor looks at the car either directly underneath or from an angle (perhaps mounted at the edge of a sidewalk), both have been tested using the preset “Ground mounted” in Exploration tool. This case was tested using a reference design casing without any integrated lens.

Testing was performed by mounting the sensor in an appropriate way and measuring for 30 seconds. Performance when a person is (when applicable) moving in front of the sensor were undertaken as well. Algorithm specific settings were optimized after this, which is also reflected in the presets found in Exploration tool.

All test cases were fully completed without any issues.

The obstruction system was tested by obstructing the sensor with different objects. The temperature tested by calibrating the sensor in ambient room temperature and then placing the sensor in a freezer as well as taking it outside during a cold winter day in Sweden. No issues with the obstruction detection were found during these tests.

Temperature

All tests have been performed outside in southern Swedish winter conditions (around 0 degrees Celsius ambient temperature) while the sensor was calibrated in indoor conditions. So a temperature difference slightly below 20 degrees was experienced without issue.

3 Memory

3.1 Flash

The reference application compiled from ref_app_parking.c on the XM125 module requires around 74 kB.

3.2 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 ref_app_parking.c.

RAM	**Size (kB)**	
Preset	*Ground*	*Pole*
Static	1	1
Heap	5	7
Stack	2	2
Total	8	10

4 Power Consumption

Average current	**Current (mA)**	
Preset	*Ground*	*Pole*
	0.075	1.075

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.