A121 Parking Reference Application User Guide

User Guide

A121 Parking Reference Application User Guide

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 Introduction

This *Reference Application* demonstrates how the A121 sensor can be used as a parking sensor. The sensor can be mounted either on the ground or on a pole, within or near a parking space. This Reference Application indicates whether a car is parked in the designated space while maintaining low power consumption. This Reference Application also includes a component called *Obstruction Detection*. The purpose of this component is to identify if something is blocking the sensor, which could interfere with parking detection.

3 Your First Measurement

In this section you will find the information needed to make your first measurement with the Parking Reference Application.

3.1 Exploration Tool

To evaluate parking detection with the Acconeer radar, we recommend that you start with one of our *Evaluation Kits* and the *Exploration Tool* application.

After the steps in Getting started have been done, press *Parking* in the left hand side menu and then press the *Calibrate* button followed by *Start measurement*. With the default settings, the radar will search for a parked car in the interval of 0.1 m to 0.4 m with an update rate of 0.1 Hz. To get a faster response from the sensor while trying out the application you can increase the update rate to, for example, 5 Hz. If you place a stationary object approximately 20 cm from the sensor, the Parking Reference Application should report it as a parked car. If you place an object straight onto or very close to the sensor the Parking Reference Application should output *Obstruction detected*.

Since the default settings have the obstruction detection activated, the GUI shows three plots, as shown in Figure 1. If the obstruction detection is deactivated, we would only see two plots in the GUI.

The main plot, called *Sampled Signatures*, shows the *signature* history and their location in the 2D space. If a car is detected, two lines indicating the range for which the object signature must be within to be considered the same object. If a large enough fraction of signatures is above the threshold and within the range, detection is triggered and indicated.

The secondary plot, called *Noise adjusted amplitude* shows the amplitude levels after normalizing against an approximation of the noise level and adjusting with the range. This is the input to each *signature* calculation. You can read more about the algorithm in the section *Algorithm Signal Processing*

The obstruction plot is only shown when the obstruction detection is activated. This plot shows the amplitude data for the region that the obstruction detection is active in as well as the signature of the displayed data as an orange point. In addition, the signature of the calibration data is also shown together with a bounding box based on the distance threshold. If the orange point moves outside the box, the obstruction detection will trigger.

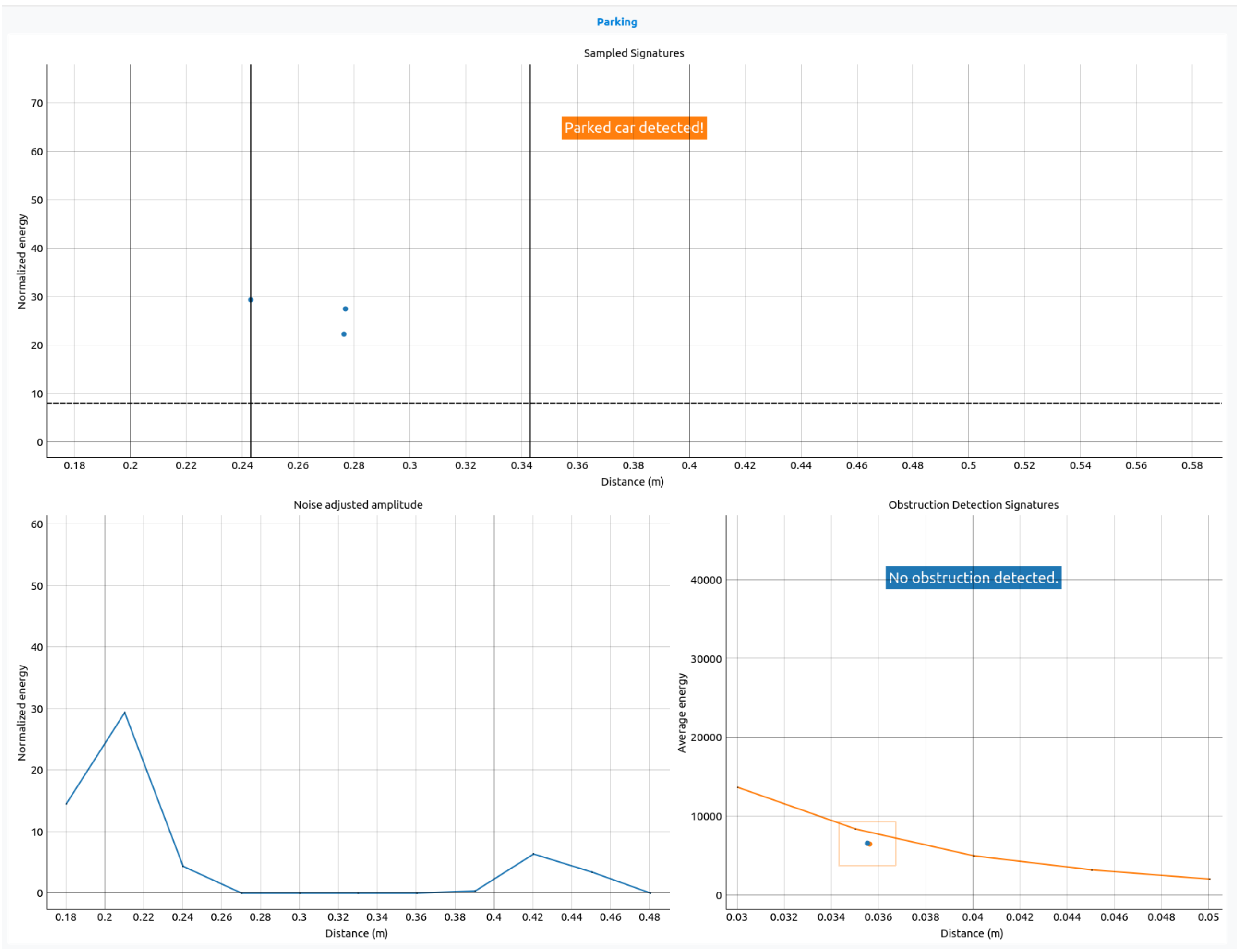

Figure 1: Example of the parking GUI with obstruction detection active. A parked car is reported and the reported object has been seen for three frames (which is indicated that you have three dots above the threshold).

3.2 Embedded C

An embedded C application is provided in the Acconeer SDK, available at the Acconeer Developer Site.

The embedded application use the same default configuration as Exploration Tool. By default, it prints the result using *printf* which usually is connected to stdout or a debug UART, depending on environment. The application is provided in source code.

4 Configuration

This section outlines how to configure the Parking Reference Application in common scenarios.

4.1 Presets

The Parking Reference Application has two predefined configurations, available in the application as presets, with the following purposes:

Ground
: This preset is suitable for scenarios where the sensor is located close to the car, typically for ground-mounted or curb-mounted parking sensors. The update rate of this preset is set to 0.1 Hz to maintain low power consumption, as many ground-mounted parking sensors are battery-powered and do not require a fast response time.

Pole
: This preset is suitable for use cases where the sensor observes the car from a longer range, typically found in pole-mounted parking sensors that view the car from the side or slightly downward. This preset features a higher update rate compared to the ground-mounted preset, as many pole-mounted parking sensors are connected to the power grid. Additionally, the higher update rate helps make the detection less sensitive to interruptions, such as when a person walks between the car and the pole.

These presets are available both in the Exploration Tool application and in our C example. They should be viewed as a starting points, from where a more tailored configuration can be developed.

4.2 Further Configuration

This section describes further configurations that can be made to tailor the application to your use case.

Setting the Measurement Range
: Adjustments to the measurement range can be done by changing the range settings (*range_start_m* and *range_end_m*). These determine the approximate range from the sensor wherein you expect to find a part of a car. Note that it is detrimental to the performance 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. This is constricted in Exploration Tool application and API.

Obstructed Sensor
: If an object is blocking the sensor, even if the start of the measurement range is set beyond it, it will affect the signal and it might also affect the sensor's ability to detect a parked car. For some use cases it is important to know if the sensor is obstructed or not and in these cases one can enable the Obstruction Detection component of the Parking Reference Application. You can read more about how this part of the algorithm works in *Obstruction Detection*.

Setting the Update Rate
: The main factor behind the energy consumption for this Reference Application is the update rate. For a battery driven application it is therefore beneficial to set the update rate as low as the use case will allow to prolong the battery life of the device.

5 Physical Integration

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 3 and Figure 2.

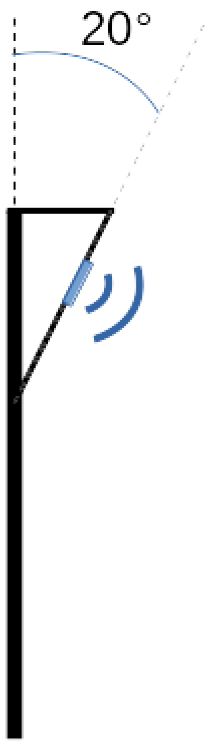

Figure 2: Illustration of the sensor mount with angle used in testing the pole mounted case.

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.

6 Calibration

There are two types of calibration to consider for this application, for normal usage, where only the parking detection part of the application is used, and an additional for when obstruction detection is used. Both calibrations are done at start-up, but the obstruction detector can require regular re-calibration.

Parking Detection

The calibration for the parking detection part of the application is done with the TX antenna turned off, so there is no dependence on what is in front of the sensor at time of calibration. It can be thought of as part of the start-up sequence without any operator requirements. The algorithm has an internal model to compensate for the temperature fluctuations in the environment which otherwise would impact the data, so after the calibration has finished, there is no need for additional calibration unless the power is dropped.

Obstruction Detection

For obstruction detection functionality, stricter calibration requirements apply. The obstruction calibration analyzes the range close to the sensor while transmitting pulses, thus no object (except a potential casing) can be present within 10 cm of the sensor during this calibration, when using the default configuration for obstruction detection. The obstruction detection range can be configured and in that case no object can be present within the obstruction detection range plus an additional margin of approximately 5 cm. In addition, optimal performance requires calibration under conditions close to normal operation, meaning that the sensor needs to be installed into its intended geometry. This is not required for the regular parking detection functionality, only for the obstruction detection. If the obstruction detector is activated and the temperature is estimated to deviate more than 20 degrees, it is recommended to re-calibrate the obstruction detector.

7 Reference Application Output

The reference application will provide one or two main output: detection and if activated, obstruction. The parking detection output will be *True* if the algorithm detects a car in the range and *False* otherwise. The obstruction detection will output *True* if the algorithm detects an obstruction of the sensor. If the obstruction detection is not activated in the configuration, this will always be *False*.

In addition, the Reference Application provides detailed information that is mainly used for plotting, but can of course be used for other purposes within your application. This result is only available in the Exploration Tool API and not in the embedded implementation for a microcontroller.

8 Algorithm Signal Processing

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 Sparse IQ data). The phase information is not used in this reference application. See Frames, sweeps and subsweeps for more information about frames and sweeps.

8.1 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.

8.2 Obstruction Detection

The Obstruction Detection component of this Reference Application measures few points very close to the sensor, within the so-called *direct leakage*. The obstruction detection feature allows the user to configure both the sensitivity and the range of the processing. Since the amplitude is sensitive to temperature fluctuations, it is recommended to test the threshold under conditions appropriate to the use case. If the obstruction detection feature is used, it is important to ensure that the range of obstruction detection does not overlap with the range for parked car detection.

8.3 Calibration

As stated in previous sections both the parking detection part of this application and the obstruction detection part needs to be calibrated. The parking detection part uses an estimation of the underlying noise level to calculate when an object is in front of it. The calibration is necessary to estimate this noise level. The noise level is mainly dependent on the temperature, so the calibration not only stores the noise level, but also the temperature measured by the sensor at the time of calibration. The algorithm automatically compensates for the change in the environment temperature which is the reason why this calibration is only necessary to do at start-up.

However, the calibration for obstruction detection requires measurements where both transmission and reception of pulses are used to characterize the *direct leakage*. Since direct leakage varies with temperature and the algorithm does not compensate for this, it is recommended to recalibrate when the temperature is estimated to deviate by more than 20 degrees from the original calibration temperature.

9 Memory and Power Consumption

In this section you can find the RAM memory usage and power consumption for the embedded C application.

9.1 Memory

The table below shows approximate RAM for the embedded C application using an XM125 for the two presets.

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

9.2 Power Consumption

The table below shows the average current in mA using an XM125 for the two presets.

Ground (mA)	Pole (mA)
0.075	1.075

10 Test Results

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 3.

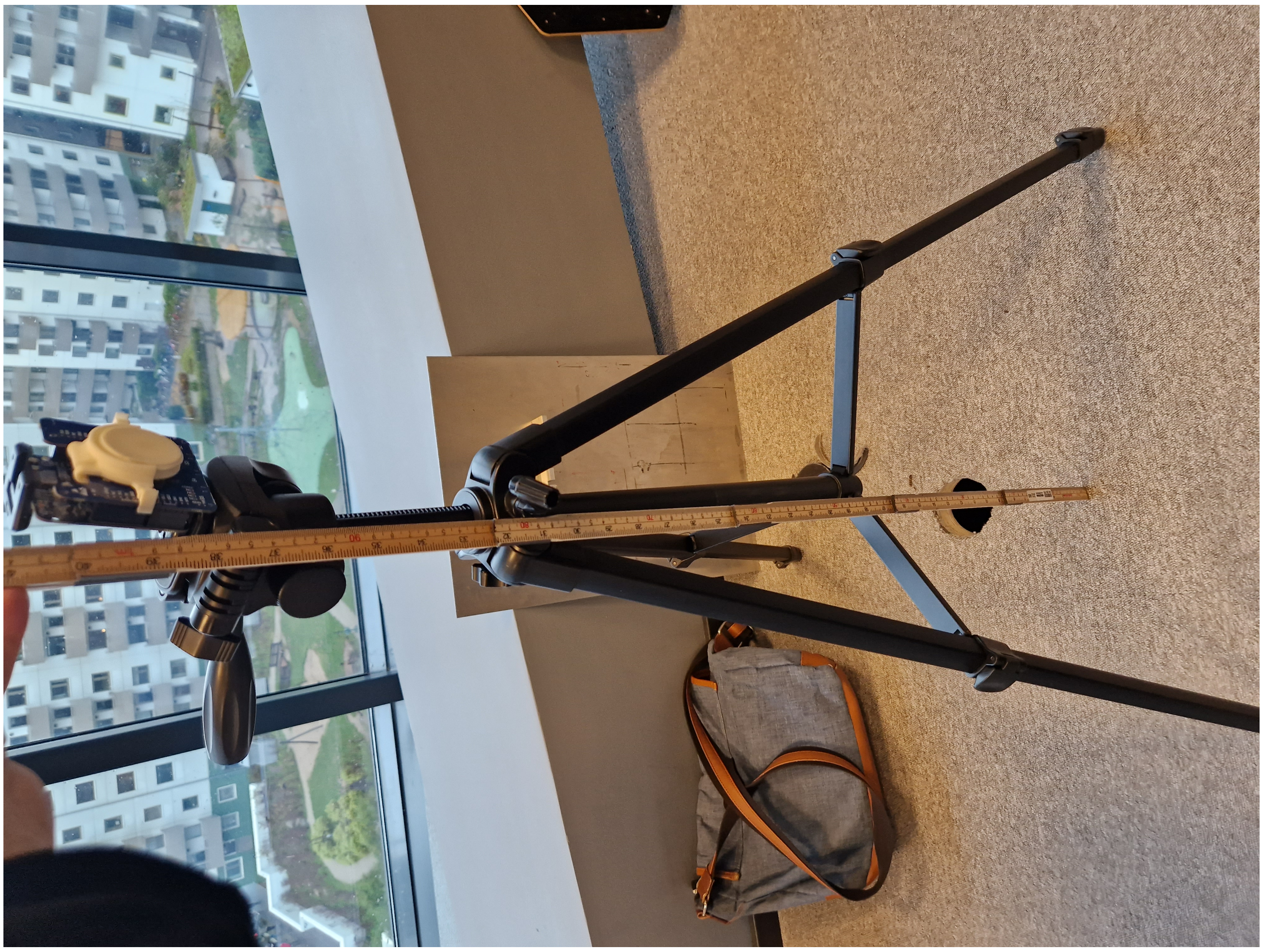

Figure 3: Test setup for the pole mounted testing, note that the sensor is tilted slightly downwards (about 20 degrees from a ground perpendicular axis). An FZP lens was also used, as seen in the picture.

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.

10.1 Temperature

All parking 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.

10.2 Obstruction Detection

The obstruction detection has been tested in a temperature oven by first calibrating in ambient (25 degrees) and then heating/cooling and performing an obstruction in regular intervals. The obstruction (under default settings) works to 22 degrees deviation, where the sensor start to report constant obstruction.

11 Disclaimer