acconeer

Distance Detector

User Guide

Distance Detector

User Guide

Author: Acconeer AB

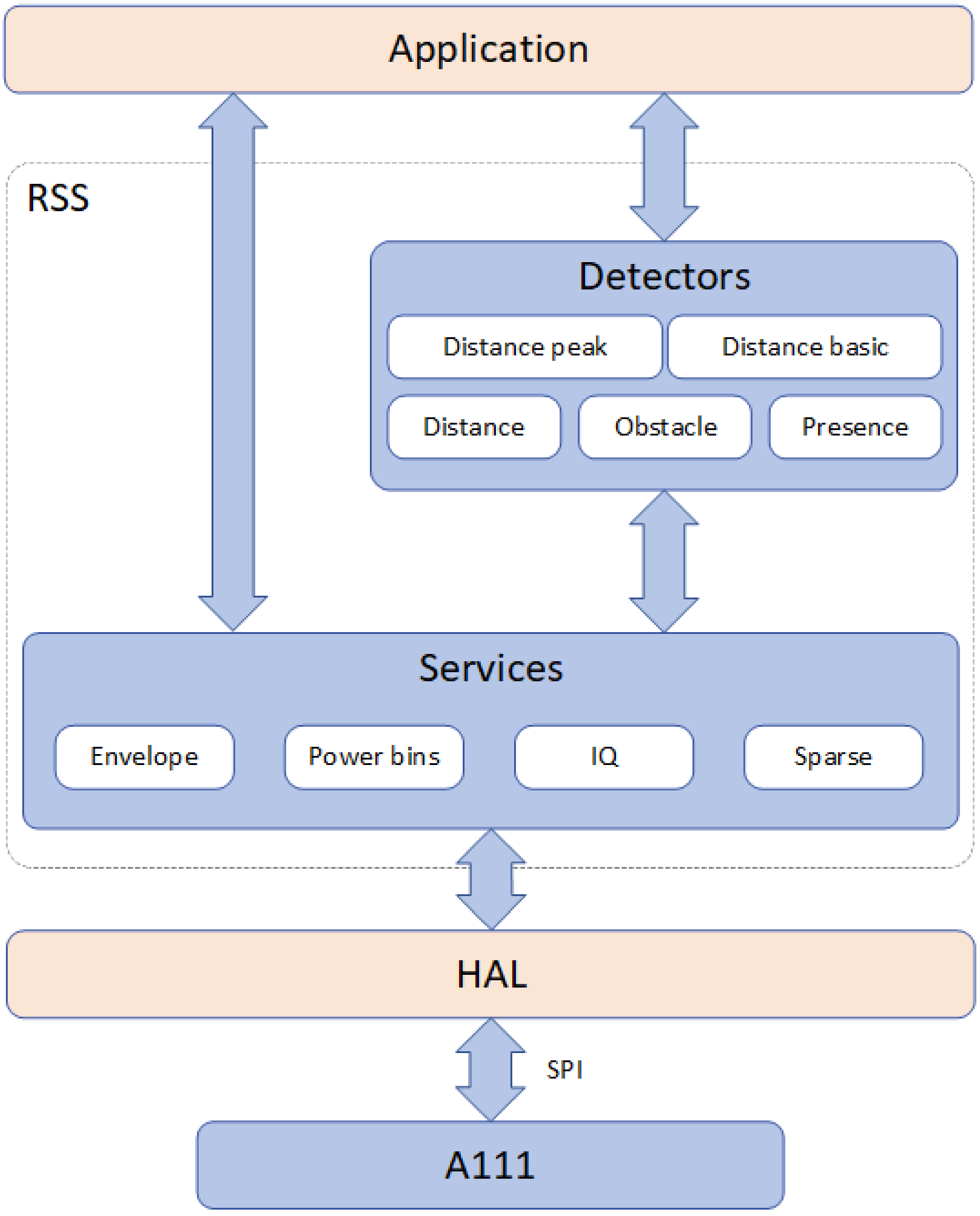
Version:a111-v2.15.2

Acconeer AB August 18, 2023

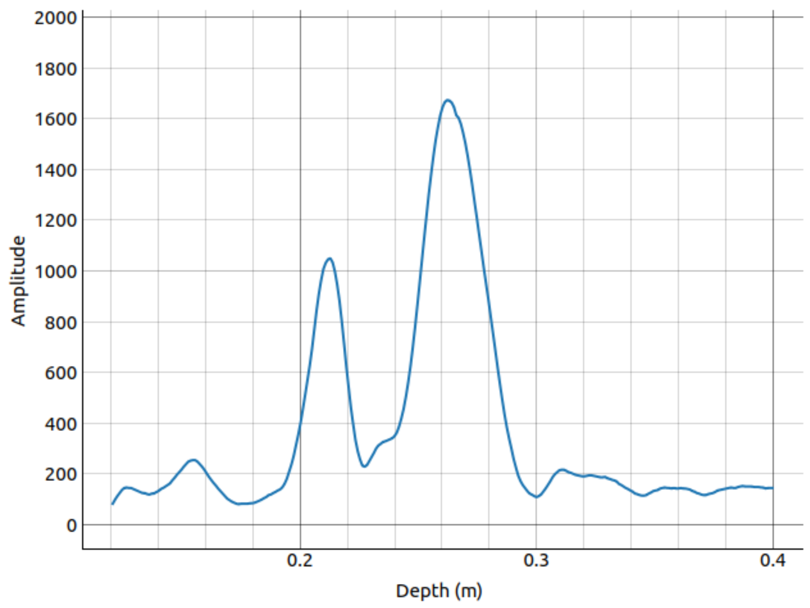
Contents

1 Distance Detector

The Distance Detector provides an API to get the distance to one or several objects in front of the sensor. It is implemented on top of the Envelope service and data from the Envelope service is further processed to find objects in the signal data.



The figure below shows envelope data with two objects in front of the sensor. The Distance Detector uses algorithms to find the peaks of the two objects and returns the distance and the amplitude of the peaks to the client application. Note that the absolute distance to an object is affected by integration of the sensor behind different material and lenses and that the amplitude is the peak value from the underlying Envelope data. For more details on the envelope data it is recommended to use our exploration tool and documentation on GitHub Acconeer Exploration Tool and also the corresponding documentation on Read-the-Docs.



Acconeer provides an example of how to use the Distance Detector: example_detector_distance.c

1.1 Initializing the System

The Radar System Software (RSS) must be activated before any other calls are done. The activation requires a pointer to an acc_hal_t struct which contains information on the hardware integration and function pointers to hardware driver functions that are needed by RSS. See chapter 4 in the document "HAL Integration User Guide" for more information on how to integrate the driver layer and populate the hal struct.

In Acconeer's example integration towards STM32 and the drivers generated by the STM32Cube tool, there is a function acc_hal_integration_get_implementation to obtain the hal struct.

```
const acc_hal_t *hal = acc_hal_integration_get_implementation();

if (!acc_rss_activate(hal))
{
    /* Handle error */
}
```

The corresponding code looks slightly different in software packages for the Raspberry Pi and other software packages from Acconeer where peripheral drivers for the host are included. The hal struct is then obtained with the function acc_hal_integration_get_implementation.

```
const acc_hal_t *hal = acc_hal_integration_get_implementation();

if (!acc_rss_activate(hal))
{
    /* Handle error */
}
```

1.2 RSS Configuration

There is one configuration for RSS that takes effect for all services and detectors. That configuration is 'Override Sensor ID Check at Creation' and makes it possible to create multiple services and/or detectors for the same sensor ID. The configuration can be set by calling:

```
acc_rss_override_sensor_id_check_at_creation(true);
```

A normal situation where this can be of benefit is when an application wants to switch between services and/or detectors easily and efficiently or when an application wants to switch between configurations of the same service/detector. An example of how to do this can be found in example_multiple_service_usage.c.

2 Configuring the Distance Detector

A configuration must be created to use the Distance Detector. To create a configuration, call the acc_detector_distance_configuration_create function which will create a configuration and return it.

```
acc_detector_distance_configuration_t distance_configuration;

distance_configuration = acc_detector_distance_configuration_create();
```

A newly created configuration is populated with default parameters and can be used directly to create the detector by calling the acc_detector_distance_create function. A more common scenario is to first modify some of the configuration parameters to better fit the application.

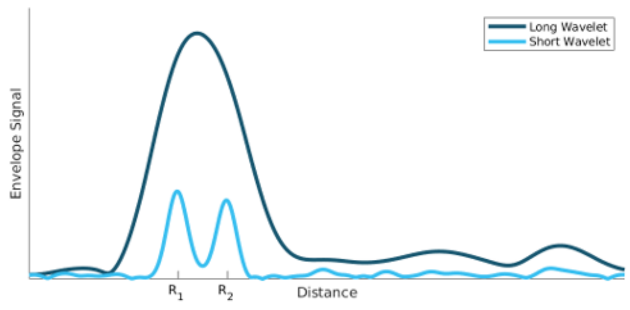
2.1 Setting Sweep Parameters

Many of the sweep parameters configurable for the Envelope service are also configurable through the Distance Detector. Like other configuration parameters, the sweep parameters have reasonable default values, but in most applications it is necessary to modify at least some of them.

See acc_detector_distance.h for more complete explanation of the configuration parameters.

2.1.1 Profiles

The services and detectors support profiles with different configuration of emission in the sensor. The different profiles provide an option to configure the pulse length and optimize on either depth resolution or radar loop gain. More information regarding profiles can be read in the Radar sensor introduction document.



The figure above shows the envelope signal of the same objects with two different profiles, one with a short pulse length and one with a long pulse length.

The Distance Detector supports five different profiles which are defined in acc_definitions_a111.h. Profile 1 has the shortest pulse length and should be used in applications which aim to see multiple objects or with short distance to the object. Profiles with higher numbers have longer pulse length and are more suitable to use in applications which aim to see objects with weak reflection or objects further away from the sensor.

The highest profiles, 4 and 5, will lead to lower precision when estimating the distance.

Profiles can be configured by the application by using a set function in the Distance Detector API. The default profile is ACC_SERVICE_PROFILE_2.

```
void acc_detector_distance_configuration_service_profile_set(
    acc_detector_distance_configuration_t distance_configuration,
    acc_service_profile_t service_profile);
```

2.1.2 Downsampling Factor

In the Distance Detector, the base step length is ˜0.5mm. The default configuration enables the sensor to produce data at every point and will give the highest resolution. Applications that don't require as high resolution can downsample the data in the sensor by increasing the step length. For example setting downsampling factor to 4 makes the distance between two points in the measured range ˜2mm. Less data require less processing and could be useful in applications which require low power and memory consumption. Downsampling will also make it possible to set a longer range length. The Distance Detector supports a downsampling factor of 1, 2, or 4.

```
void acc_detector_distance_configuration_downsampling_factor_set(
    acc_detector_distance_configuration_t distance_configuration, uint16_t
    downsampling_factor);
```

2.1.3 Hardware Accelerated Average Samples (HWAAS)

The sensor can be configured with the number of samples measured and averaged to obtain a single point in the data. These samples are averaged directly in the sensor hardware and only one value for each point is transferred over SPI. Therefore, increasing HWAAS is a both memory and computationally inexpensive way to increase the SNR. The time needed to measure a sweep is roughly proportional to the number of averaged samples. Hence, if there is a need to obtain a higher update rate, HWAAS could be decreased but this leads to lower SNR. The HWAAS value must be at least 1 and not larger than 63, the default value for the Distance Detector is 10.

```
void acc_detector_distance_configuration_hw_accelerated_average_samples_set(
    acc_service_configuration_t configuration, uint8_t samples);
```

2.1.4 Power Save Mode

The power save mode configuration sets what state the sensor waits in between measurements in an active service. There are five power save modes and the modes differentiate in current dissipation and response latency, where the most current consuming mode 'ACTIVE' gives fastest response and the least current consuming mode 'OFF' gives the slowest response. The absolute response time is determined by several factors besides the power save mode configuration. These are length and threshold. In addition, the host capabilities in terms of SPI communication speed and processing speed also impact on the absolute response time. The power consumption of the system depends on the actual configuration of the application and it is recommended to test both the minimum response time and the power consumption when the configuration is decided.

Mode 'HIBERNATE' is not supported by the Distance Detector.

```
typedef enum
{
    ACC_POWER_SAVE_MODE_OFF,
    ACC_POWER_SAVE_MODE_SLEEP,
    ACC_POWER_SAVE_MODE_READY,
    ACC_POWER_SAVE_MODE_ACTIVE,
    ACC_POWER_SAVE_MODE_HIBERNATE,
} acc_power_save_mode_enum_t;
typedef uint32_t acc_power_save_mode_t;
```

```
void acc_detector_distance_configuration_power_save_mode_set(
    acc_detector_distance_configuration_t distance_configuration,
    acc_power_save_mode_t                 power_save_mode);
```

2.1.5 Maximum Unambiguous Range (mur)

Sets the maximum unambiguous range (MUR), which in turn sets the maximum measurable distance (MMD).

The MMD is the maximum value for the range end, i.e., the range start + length. The MMD is smaller than the MUR due to hardware limitations.

The MUR is the maximum distance at which an object can be located to guarantee that its reflection corresponds to the most recent transmitted pulse. Objects farther away than the MUR may fold into the measured range. For example, with a MUR of 10 m, an object at 12 m could become visible at 2 m.

A higher setting gives a larger MUR/MMD, but comes at a cost of increasing the measurement time for a sweep. The measurement time is approximately proportional to the MUR.

This setting changes the pulse repetition frequency (PRF) of the radar system. The relation between PRF and MUR is MUR = c / (2 * PRF) where c is the speed of light.

Setting	MUR	MMD	PRF
ACC_SERVICE_MUR_6	11.5 m	7.0 m	13.0 MHz
ACC_SERVICE_MUR_9	17.3 m	12.7 m	8.7 MHz

This is an experimental feature.

```
void acc_detector_distance_configuration_mur_set(
    acc_detector_distance_configuration_t distance_configuration,
                                                             acc_service_mur_t

                                                                 max_unambiguous_range);
```

2.2 Distance Detector parameters

Distance Detector parameters controls the post processing of the data provided by the Envelope service. These settings will help to improve the service data and control how peaks are found and sorted.

For more information on detector specific parameters see Distance Detector Processing document.

2.2.1 Sweep Averaging

With sweep averaging, multiple envelope sweeps can be averaged in the detector. This will reduce the impact of noise and enables a more robust detector by lowering the false alarm rate with the same probability of detection. Setting sweep averaging > 1 will enable sweep averaging and also increase memory consumption. As default sweep averaging is 5.

```
void acc_detector_distance_configuration_sweep_averaging_set(
    acc_detector_distance_configuration_t distance_configuration, uint16_t
                                    sweep_averaging);
```

2.2.2 Threshold type

The Distance Detector supports three kinds of thresholds. Fixed Threshold is the default threshold type.

Threshold type	Purpose	Memory usage	Complexity
Fixed Threshold	Simple	Low	Low
Recorded Threshold	Remove static peaks	High	Medium
Constant False Alarm Rate (CFAR) Threshold	Dynamic	Low	High

```
void acc_detector_distance_configuration_threshold_type_set(
    acc_detector_distance_configuration_t  distance_configuration,
    acc_detector_distance_threshold_type_t threshold);
```

Fixed Threshold Fixed Threshold sets a fixed threshold over the full range. Peaks with an amplitude above the threshold will be reported to the application

```
void acc_detector_distance_configuration_fixed_threshold_set(
    acc_detector_distance_configuration_t distance_configuration, uint16_t
    threshold);
```

Recorded Threshold Recorded Threshold will record the background and calculate a threshold. This will filter out any stationary objects in front of the sensor and only report distance to objects not present during the recording of the background. The Distance Detector has to be provided sufficient memory to store the background data in. The required length of the background data is provided by the acc_detector_distance_metadata_get function.

```
bool acc_detector_distance_record_background(acc_detector_distance_handle_t
    distance_handle, uint16_t *background, uint16_t background_length,
    acc_detector_distance_recorded_background_info_t *background_info);
```

If the environment surrounding the sensor significantly changes, then a new background recording should be done. This is done by calling the acc_detector_distance_record_background function.

A higher number of background sweeps will result in a more stable background, but will take a longer time to record.

```
void acc_detector_distance_configuration_record_background_sweeps_set(
    acc_detector_distance_configuration_t distance_configuration, uint16_t
    record_sweeps);
```

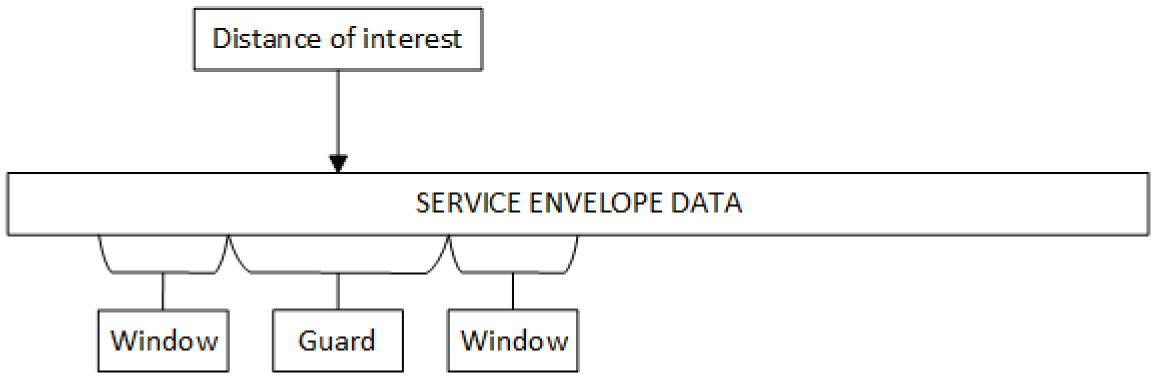
The threshold is also configurable by setting the sensitivity, a number between 0 and 1. A higher value will make the detector more sensitive and increase the detection rate, but will also increase the number of false detects.

```
void acc_detector_distance_configuration_threshold_sensitivity_set(
    acc_detector_distance_configuration_t distance_configuration, float
    sensitivity);
```

The recorded background can be saved and reused by using the acc_detector_distance_set_background function.

```
bool acc_detector_distance_set_background(acc_detector_distance_handle_t
    distance_handle, const uint16_t *background, uint16_t background_length);
```

Constant False Alarm Rate (CFAR) Threshold CFAR Threshold constructs a threshold for a certain distance by using the envelope signal from neighboring distances in the same sweep. This provides a more dynamic threshold than Fixed Threshold while keeping memory consumption down.



The parameters configuring the CFAR Threshold are the guard, the gap around the distance of interest which are not included in the sweep, and the window, the distance on either side of the guard included in the threshold calculation.

```
void acc_detector_distance_configuration_cfar_threshold_guard_set(
    acc_detector_distance_configuration_t distance_configuration, float
    guard_m);
void acc_detector_distance_configuration_cfar_threshold_window_set(
    acc_detector_distance_configuration_t distance_configuration, float
    window_m);
```

Instead of determining the CFAR Threshold from sweep amplitudes on both sides of the distance of interest, it may be useful to only use the window closer to the sensor. This is useful in cases when many multipath signals can appear just after the main peak, e.g. fluid level in small tanks.

```
void
    acc_detector_distance_configuration_cfar_threshold_only_lower_distance_set
    (acc_detector_distance_configuration_t distance_configuration, bool
    only_lower_distance);
```

The sensitivity of the threshold can be set in the same way as for the Recorded Threshold.

2.2.3 Peak sorting

When detecting multiple peaks, one peak might be more important than the others depending on use-case. The Distance Detector supports four different peak sorting methods. Strongest First is the default peak sorting.

Closest First This method sorts the peaks by distance, with the closest first.

Strongest First This method sorts the peaks by amplitude, with the strongest first.

Strongest Reflector First Objects at a larger distance will have a lower amplitude. This method takes into account the radar equation and will sort the peaks by $A_i*R_i^2$, where A_i is the amplitude of the peak and R_i is the distance to the peak.

Strongest Flat Reflector First Large flat reflectors, such as fluid surfaces, will have a different distance dependence in the radar equation, compared to other reflectors. This method will take into account the radar equation and sort peaks by A_i*R_i, where A_i is the amplitude of the peak and R_i is the distance to the peak.

3 Measure Distances

3.1 Creating and Activating the Distance Detector

When the configuration has been prepared with the desired configuration parameters, the Distance Detector can be created. During the creation step all configuration parameters are validated and the resources needed by RSS are reserved. This means that if the creation step is successful, we can be sure that it is possible to activate the detector and get data from the sensor (unless there is some unexpected hardware error).

```
acc_detector_distance_handle_t distance_handle =
   acc_detector_distance_create(distance_configuration);
```

When the detector is created, it is possible to fetch metadata from the detector

```
acc_detector_distance_metadata_t   metadata;

acc_detector_distance_metadata_get(distance_handle, &metadata);

float    start_m           = metadata.start_m;
float    length_m          = metadata.length_m;
uint16_t background_length = metadata.background_length;
```

When the Distance Detector has been created, the memory resources allocated by the configuration can be released by calling the acc_detector_distance_configuration_destroy function.

To activate the detector call the acc_detector_distance_activate function. Now, the detector and the sensor are ready to produce data which can be retrieved by calling the acc_detector_distance_get_next function.

```
if (!acc_detector_distance_activate(distance_handle)) {
    /* Handle error */
}
```

3.2 Getting Detection Results

After activation it is possible to retrieve the distance results by calling the acc_detector_distance_get_next function.

```
uint16_t                              number_of_peaks = 5;
acc_detector_distance_result_t        result[number_of_peaks];
acc_detector_distance_result_info_t   result_info;

if (!acc_detector_distance_get_next(distance_handle, result, number_of_peaks
   , &result_info))
{
    /* Handle error */
}

for (uint16_t i = 0; i < result_info.number_of_peaks; i++)
{
    printf("Amplitude %u at %u mm\n", (unsigned int)result[i].amplitude,
           (unsigned int)(result[i].distance_m * 1000));
}
```

4 Deactivating and Destroying the Distance Detector

To release the memory resources allocated by the Distance Detector, call the acc_detector_distance_deactivate function followed by the acc_detector_distance_destroy function. The Distance Detector can be activated again after deactivate, but it has to be recreated after destroy.

```
if(!acc_detector_distance_deactivate(distance_handle))
{
    /* Handle error */
}
acc_detector_distance_destroy(&distance_handle);
```

5 Updating from Distance Peak Detector

The Distance Detector is replacing the Distance Peak Detector. The API is similar, except for some additional functionality added to the Distance Detector, and it should be easy to upgrade an application to use the Distance Detector.

5.1 Configuration

The Distance Detector does not expose acc_base_configuration_t and is instead configured directly through acc_detector_distance_configuration_t. Most settings are the same and are configured the same way. The default service profile is changed from ACC_SERVICE_PROFILE_1 to ACC_SERVICE_PROFILE_2.

Instead of configuring *running average factor*, the Distance Detector introduces *sweep averaging*.

Configuring threshold has changed and an additional threshold type is introduced. *Fixed* threshold type is kept with the same functionality. Threshold modes *estimation* and *provided* is replaced by *recorded*. Recorded threshold can be used both to record the background and to provide an already recorded background to the detector. The additional threshold mode is Constant False Alarm Rate, CFAR (see above for more information on this threshold).

The option to set *absolute amplitude* is removed and the Distance Detector always returns the absolute amplitude.

Additional peak sorting methods have been introduced and the API has changed. *Sort by amplitude* corresponds to ACC_DETECTOR_DISTANCE_PEAK_SORTING_STRONGEST_FIRST and the default peak sorting for the Distance Peak Detector corresponds to ACC_DETECTOR_DISTANCE_PEAK_SORTING_CLOSEST_FIRST.

5.2 Getting Detection Result

The API for getting the detection result is similar for the Distance Peak Detector and the Distance Detector.

The acc_detector_distance_result_info_t contains more information. The number of peaks returned is moved to the result info. Also, the Distance Detector returns information about the first sample point above the threshold.

6 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.