Presence Detector

User Guide

Presence Detector

User Guide

Author: Acconeer AB

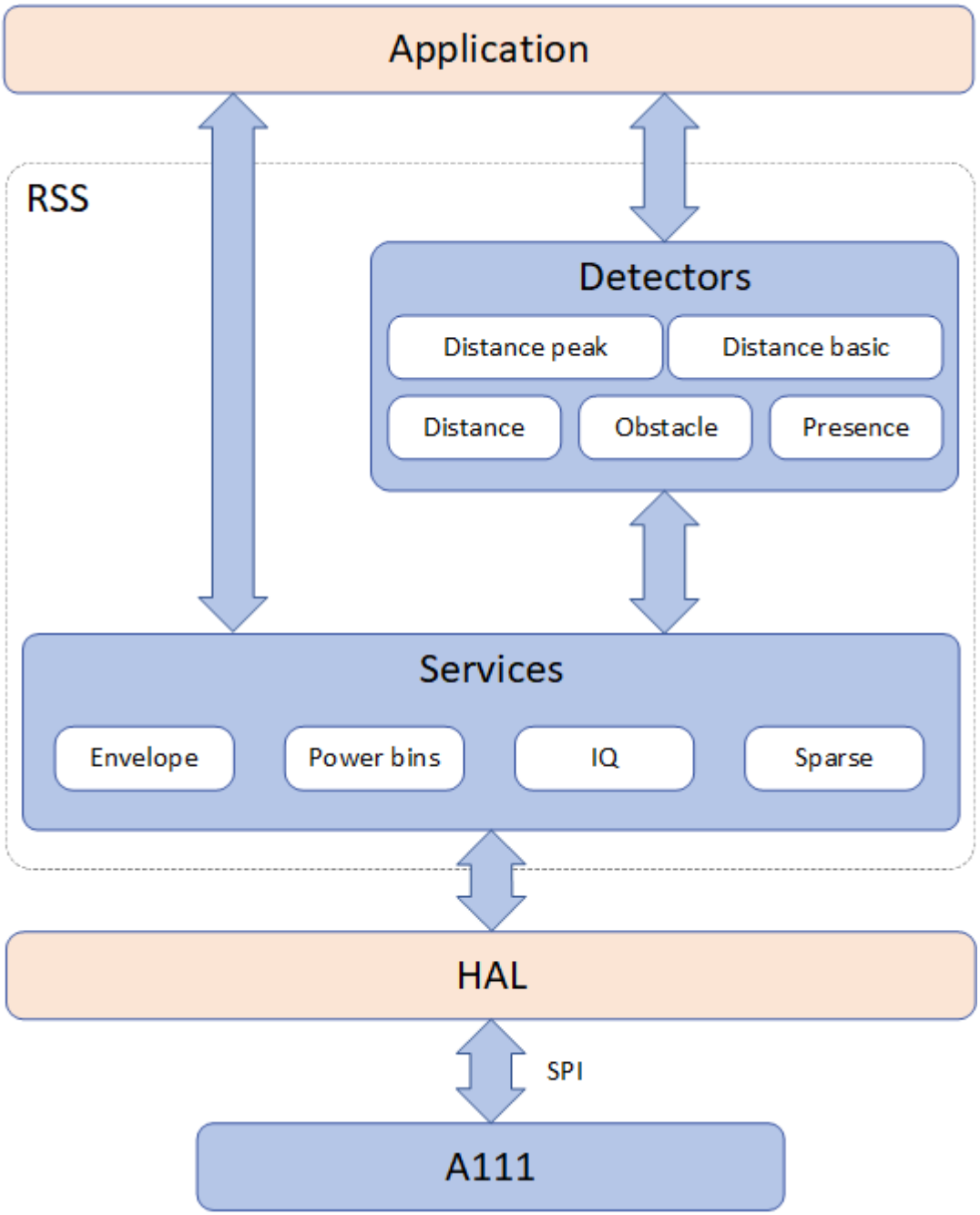
Version:a111-v2.15.2

Acconeer AB August 18, 2023

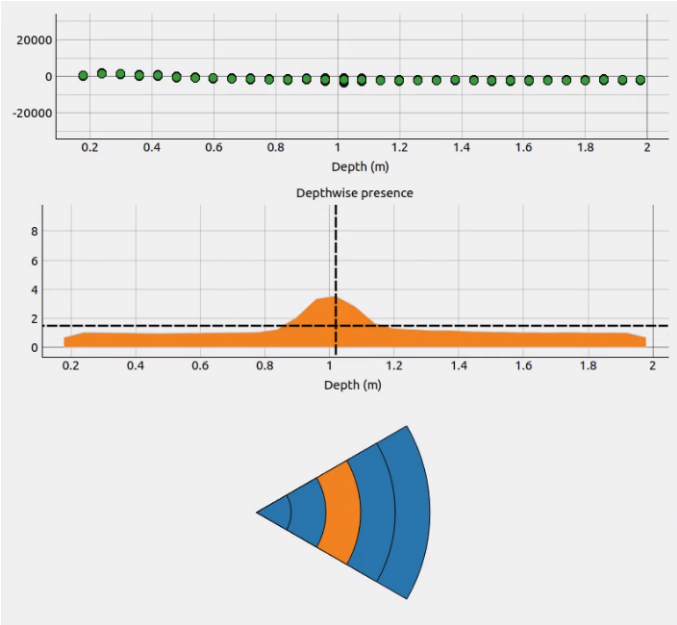
Contents

1 Presence Detector

The presence detector provide software to detect movement in front of the sensor, from slow movement like a person resting in a sofa to fast movement like a person walking towards the sensor. The detector is based on the Sparse service and utilize the public api.



The detector is designed to detect movement with a low update rate to enable low power use cases as well as higher update rate to keep track of a person walking within the range of the sensor. The detector provides output data to detect movement as well as distance to where the movement occurred. This enable applications to implement smart presence algorithms and react on movement in different areas.



For more details on the Presence algorithm and the Sparse service it is recommended to use our Exploration Tool. Check it out on GitHub Acconeer Exploration Tool.

Acconeer provides an example of how to use the presence detector: example_detector_presence.c

2 Setting up the Detector

All services and detectors in the Acconeer API are created and activated in two distinct steps. In the first creation step the configuration settings are evaluated and all necessary resources are allocated. If there is some error in the configuration or if there are not enough resources in the system to run the detector, the creation step will fail. However, when the creation is successful you can be sure that the second activation step will not fail due to any configuration or resource issues. When the detector is activated the radar is activated and can start producing data.

2.1 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 */
}
```

2.2 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.

2.3 Presence Configuration

Before the presence detector can be created and activated, we must prepare a detector configuration. First a configuration is created.

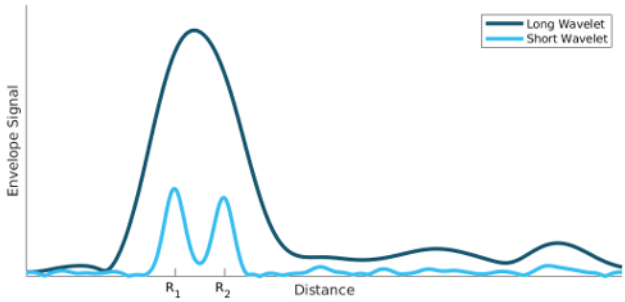
```
acc_detector_presence_configuration_t presence_configuration =
    acc_detector_presence_configuration_create();

if (presence_configuration == NULL)
{
    /* Handle error */
}
```

The newly created configuration contains default settings for all configuration parameters and can be passed directly to the acc_detector_presence_create function. However, in most scenarios there is a need to change at least some of the configuration parameters.

Configuration parameters are described in acc_detector_presence.h. This user guide explains how to use the filter parameters based on use cases to detect typical fast and slow movements. It is recommended to use Exploration Tool to find settings according to the specific use case and then transfer it to an embedded application.

2.3.1 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 and one with a long pulse.

The presence detector supports 5 different profiles which are defined in acc_definitions_a111.h. Profile 1 has the shortest pulse 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 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, are optimized for maximum radar loop gain which leads to lower precision in the distance estimate.

Profiles can be configured by the application by using a set function in the detector api. The default profile is ACC_SERVICE_PROFILE_2.

```
void acc_detector_presence_configuration_service_profile_set(
    acc_detector_presence_configuration_t configuration,
```

```
                                                                    acc_service_profile_t


                                                                        service_profile
                                                                        );
```

2.3.2 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 Presence detector is 10.

```
void acc_detector_presence_configuration_hw_accelerated_average_samples_set(
   acc_detector_presence_configuration_t configuration, uint8_t samples);
```

2.3.3 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 and also maximum update rate is determined by several factors besides the power save mode configuration. These are length, hardware accelerated average samples, sweeps per frame and sweep rate. 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 maximum update rate and power consumption when the configuration is decided.

Mode 'HIBERNATE' means that the sensor is still powered but the internal oscillator is turned off and the application needs to clock the sensor by toggling a GPIO a pre-defined number of times to enter and exit this mode. Mode 'HIBERNATE' is currently only supported by the Sparse service and require additional functions to be implemented in the HAL.

```
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_presence_configuration_power_save_mode_set(
   acc_detector_presence_configuration_t configuration,
                                                                    acc_power_save_mode_t


                                                                        power_save_mode
                                                                        );
```

The achievable update rate and power consumption of the sensor in different use cases vary between different hosts. The computational capacity and data transfer rate over SPI impacts when different modes are used in the most optimal way. A few common use cases are:

- Update rate less than 1 Hz: Mode 'OFF' turns off the sensor between sweeps and is typically used in applications which require low update rate.
- Update rate 1-4 Hz: Mode 'OFF' turns off the sensor between sweeps and should be used in applications with low update rate. In mode 'OFF', the sensor needs to be restarted and the sensor firmware loaded between updates and this have a penalty for hosts with lower SPI frequency. Therefore, it's recommended to measure the power consumption of the system with different power save modes and choose the most optimal settings when reaching update rates of 5-10 Hz.

- Update rate more than 5 Hz: Power mode ‘SLEEP’ is recommended for applications where the power consumption is important. If expected update rate is not enough with mode ‘SLEEP’, the application should use ‘READY’ instead.
- Max update rate: Select power save mode ‘ACTIVE’ for applications without power constraints that needs to maximize the update rate.

2.4 PCA based noise reduction

Strong static reflectors, such as concrete floor and metal objects, in the FoV of the radar can give higher detection level than the standard noise floor. This can cause false presence detection.

Principal component analysis(PCA) based noise reduction suppress detection from static objects while signals from real movements are preserved.

For maximum noise reduction, nbr_removed_pc is set to 2 and if the parameter is set to 0, no PCA based noise reduction is performed.

```
void acc_detector_presence_configuration_nbr_removed_pc_set(
    acc_detector_presence_configuration_t configuration, uint8_t
    nbr_removed_pc);
```

Heap usage is increased significantly when nbr_removed_pc is set to > 0. Power consumption on Cortex-M0 targets is also increased by roughly a factor of 2 when nbr_removed_pc = 2

For more information on the PCA based noise reduction algorithm, see Read-the-Docs

2.5 Creating Detector

After the presence detector configuration has been prepared and populated with desired configuration parameters, the actual detector instance must 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_presence_handle_t handle = acc_detector_presence_create(
    presence_configuration);

if (handle == NULL)
{
    /* Handle error */
}
```

If the detector handle returned from acc_detector_presence_create is equal to NULL, then some setting in the configuration made it impossible for the system to create the detector. One common reason is that the requested sweep length is too long for the selected profile, but in general, looking for error messages in the log is the best way to find out why a detector creation failed.

The configuration can be destroyed when the detector has been created

```
acc_detector_presence_configuration_destroy(&presence_configuration);
```

It is now also possible to activate the detector. This means that the radar sensor may start to produce data

```
bool status = acc_detector_presence_activate(handle);
```

2.6 Get Presence Detection

Presence data is read from the detector by a to call to the function acc_detector_presence_get_next. This function blocks until the next sweep arrives from the sensor and the result is available in result.

```
acc_detector_presence_result_t result;

for (int i = 0; i < 10; i++)
{
    status = acc_detector_presence_get_next(handle, &result);
    if (!status)
```

```
    {
        /* Handle error */
    }
}
```

2.7 Reconfigure

An application might want to change configuration at different points. For example low update rate when waiting for movement and higher update rate to track movement. The detector provides a function to reconfigure the detector.

```
if (!acc_detector_presence_reconfigure(&handle, presence_configuration))
{
    /* Handle error */
}
```

Note that a reconfiguration will destroy and setup the sparse service with the new configuration. This operation involves deallocation and allocation of memory. The current filter state will be kept if the sweep range is unchanged, otherwise it will be reset.

2.8 Deactivating and Destroying the Detector

Call the acc_detector_presence_deactivate function to stop measurements.

```
status = acc_detector_presence_deactivate(handle);
if (!status)
{
    /* Handle error */
}
```

After the detector has been deactivated it can be activated again to resume measurements or it can be destroyed to free up the resources associated with the detector handle.

```
acc_detector_presence_destroy(&handle);
```

Finally, call acc_rss_deactivate when the application doesn't need to access the Radar System Software anymore. This releases any remaining resources allocated in RSS.

```
acc_rss_deactivate();
```

2.9 Presence Result

Presence result is provided as output from get_next() function.

```
acc_detector_presence_result_t result;
acc_detector_presence_get_next(handle, &result);
```

One member variable gives an indication of presence or no presence. It's also possible to get more details about the movement that is detected, see acc_detector_presence_result_t for more information.

3 The Presence Data

3.1 Detecting Slower Movements – Inter-frame Deviation

For every frame and depth, we take the mean sweep and feed it through a fast and a slow low pass filter.

The presence detection algorithm achieves this by depthwise looking at the deviation between a fast and a slow low pass filtered version of the signal. This deviation is then filtered again both in time and depth. To be more robust against changing environments and variations between sensors, a normalization is done against the noise floor.

The inter-frame deviation is based on the deviation between the two filters.

3.2 Detecting Faster Movements – Intra-frame Deviation

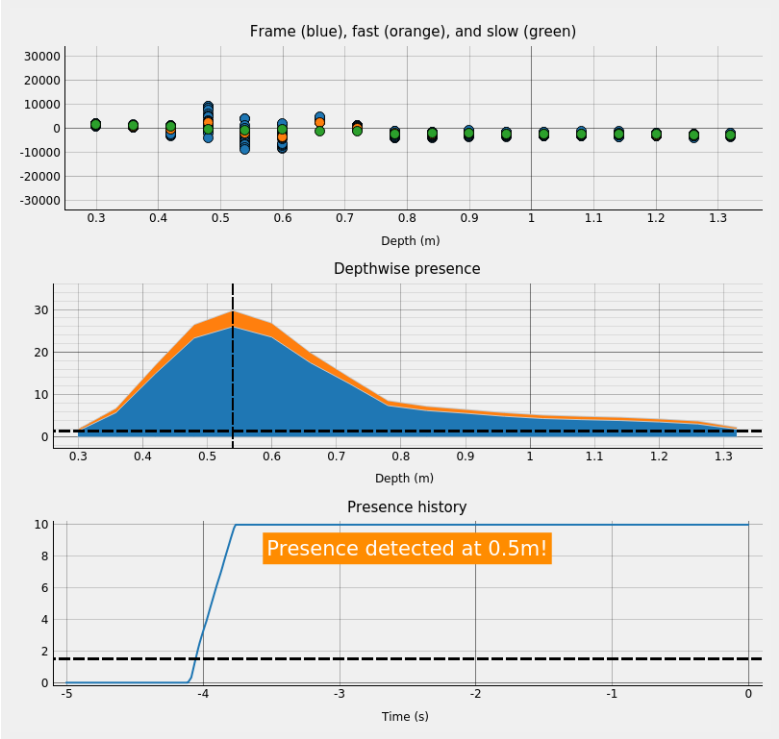
For every frame and depth, the intra-frame devition is based on the deviation from the mean of the sweeps.

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, a normalization is done against the noise floor. Finally, some simple processing is applied to generate the final output.

As previously mentioned, the inter-frame part is good at detecting slower movements, and the intra-frame part is good at detecting faster movements. By slower movements we mean, for example, a person sitting in a chair or sofa. Faster movements could be a person walking or waving their hand.

3.3 Example from Exploration Tool

The figure below provide an overview of the signals recieved by Exploration Tool.



- Top plot: The frame (blue) along with the fast (orange) and slow (green) filtered mean sweep. The distance between the fast (orange) and slow (green) dots is the basis of the inter-frame part, and the spread of the sweeps (blue) is the basis of the intra-frame part.
- Middle plot: The “depthwise presence”. This signal is the time filtered, depth filtered, and normalized version of the weighted sum of the inter- and intra-frame parts. The blue and orange parts show the inter- and intra-frame contributions respectively.
- Bottom plot: The detector output. This is obtained from taking the maximum in the above plot and low pass filtering it. The plot is limited to give a clearer view.

3.4 Common Use Cases

By default, the detector is configured such that both faster and slower movements are detected. This means that both the inter- and intra-frame parts are used. We recommend this as a starting point.

The overall sensitivity can be adjusted with the detection_threshold parameter. If the detection toggles too often, try increasing the output_time_const parameter. If it is too sluggish, try decreasing it instead.

Tuning the other parameters is described in the following sections.

3.4.1 Fast Movement

Fast movements are typically looking for a person walking towards or away from the sensor

- Disable the inter-frame part by setting the intra_frame_weight to 1

- intra_frame_time_const - Look at the depthwise presence (middle plot). If it can't keep up with the movements, try decreasing the time constant. Instead, if it's too flickery, try increasing the time constant. This will also be seen in the presence distance.

Since the inter-frame part is disabled, inter-frame parameters have no effect.

3.4.2 Slow Movement

Slow motions are typcally looking for a person resting in a sofa

- Disable the intra-frame part by setting the intra_frame_weight to 0.
- inter_frame_fast_cutoff - If too low, some (too fast) motions might not be detected. If too high, unnecessary noise might be entered into the detector. Values larger than half the update_rate disables this filter. If that is not enough, you need a higher update_rate or use intra-frame part.
- inter_frame_slow_cutoff - If too high, some (too slow) motions might not be detected. If too low, unnecessary noise might be entered into the detector, and changes to the static environment takes a long time to adjust to.
- inter_frame_deviation_time_const - This behaves in the same way as the intra-frame time constant. Look at the depthwise presence (middle plot). If it can't keep up with movements changing depth, try decreasing the time constant. Instead, if it's too flickery, try increasing the time constant.

Since the intra-frame part is disabled, the intra_frame_time_const has no effect.

4 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.