Distance Peak Detector

User Guide

Distance Peak Detector

User Guide

Author: Acconeer AB

Version:v2.0.0

Acconeer AB December 2, 2019

Contents

1 Distance Peak Detector

The Distance Peak 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 the peak of the signal.

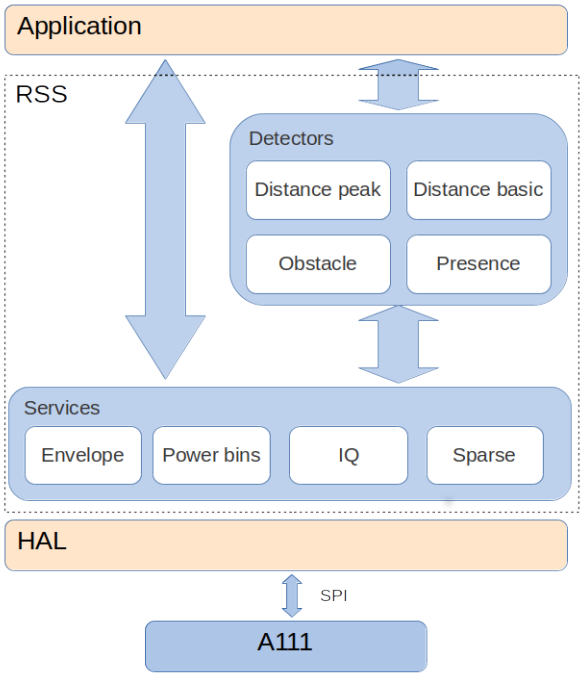

Figure 1:

The figure below show envelope data with two objects in front of the sensor. The Distance Peak detector use algorithms to find the peaks of the two objects and return distance and amplitude to the client. Note that the absolute distance to an object is affected by integration of the sensor behind different material and lenses. For more details on the envelope data it is recommended to use our exploration tool. Check it out on github, https://github.com/acconeer/acconeer-python-exploration, https://github.com/acconeer/acconeer-python-exploration.

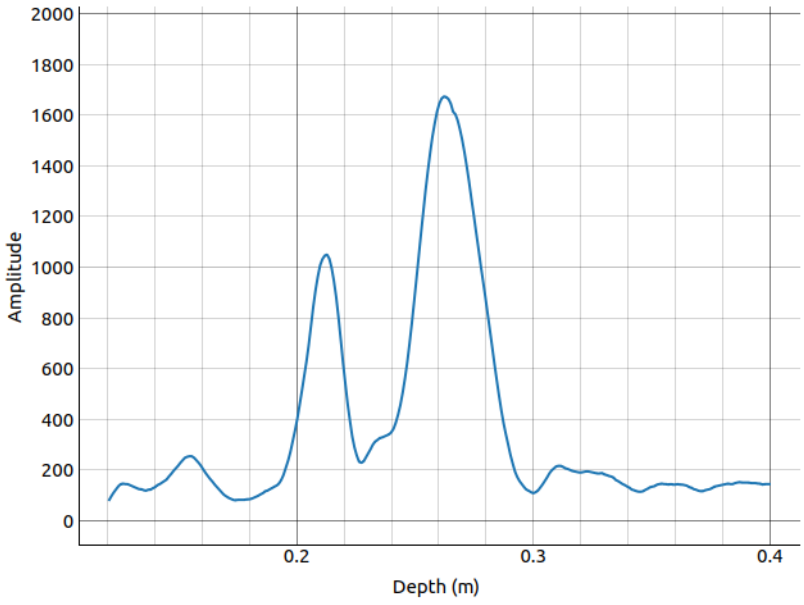

Figure 2:

Acconeer provides an example of how to use the envelope service: example_detector_distance_peak.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 to 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.

```
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 driver layer is first initialized by calling acc_driver_hal_init. The hal struct is then obtained with the function acc_driver_hal_get_implementation.

```
if (!acc_driver_hal_init())
{
    /* Handle error */
}

acc_hal_t hal = acc_driver_hal_get_implementation();

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

2 Configuring the Distance Peak Detector

To use the Distance Peak Detector, first a configuration must be created. To create a configuration, call the acc_detector_distance_peak_configuration_create function which will create a configuration and return it.

```
acc_detector_distance_peak_configuration_t distance_configuration;

distance_configuration = acc_detector_distance_peak_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_peak_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

The base configuration parameters determine the sensor id and how the sweep data will be generated in the sensor. The base configuration parameters are common to all services and are also possible to set in the Distance Peak 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. To do this we must first obtain a base configuration handle.

```
acc_base_configuration_t base_configuration;

base_configuration = acc_detector_distance_peak_get_base_configuration(
    distance_configuration);
```

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

Using the base configuration handle, we can use functions to set individual configuration parameters such as the sweep range and frequency.

```
// Set sweep start and length
acc_base_configuration_requested_start_set(base_configuration, 0.2);
acc_base_configuration_requested_length_set(base_configuration, 0.4);
```

See acc_detector_distance_peak.h and acc_base_configuration.h for a 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 wavelet length and optimize on either depth resolution or radar loop gain. More information regarding profiles can be read in the sensor introduction document.

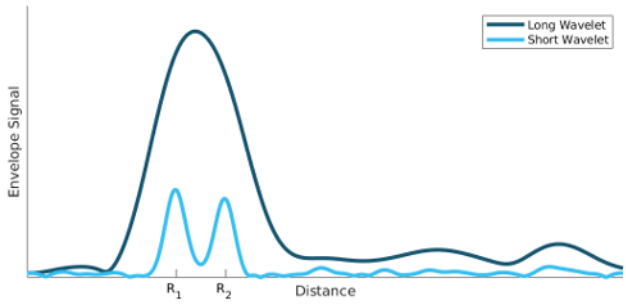

Figure 3:

The figure above shows the envelope signal of the same objects with two different profiles, one with short wavelet and one with longer.

The distance peak detector supports 5 different profiles which are defined in acc_service.h. Profile 1 has the shortest wavelet 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 wavelet 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_distance_peak_service_profile_set(
   acc_detector_distance_peak_configuration_t configuration,
                                                    acc_service_profile_t

                                                       service_profile);
```

2.2 Absolute Amplitude

The amplitude values returned by the Distance Peak Detector constitute the difference between the reflection amplitude and the threshold. The acc_detector_distance_peak_set_absolute_amplitude function can be called to configure the Distance Peak Detector to return absolute amplitude values.

```
acc_detector_distance_peak_status_t detector_status;

acc_detector_distance_peak_configuration_t distance_configuration;
```

```
distance_configuration = acc_detector_distance_peak_configuration_create();

acc_detector_distance_peak_set_absolute_amplitude(distance_configuration,
    true);
```

2.3 Adjusting the Running Average for Better Accuracy

The range and accuracy of distance measurements can be improved when running the detector using an average of multiple sweeps. This procedure may be controlled by calling the function acc_detector_distance_peak_running_average_factor_set. By setting the "factor" parameter to a value between 0-1 where 0 means that averaging is disabled. A factor 1 means that the most recent sweep has no effect on the result, which will result in that the first sweep is forever received as the result.

The current default value for this setting is 0.7. When measuring objects in motion this value may be decreased. To improve SNR for static objects the running average factor could be increased to a value closer to 1.

```
acc_detector_distance_peak_configuration_t distance_configuration;
float factor = 0.9f;

distance_configuration = acc_detector_distance_peak_configuration_create();

acc_detector_distance_peak_running_average_factor_set(distance_configuration
    , factor);
```

2.4 Threshold Mode

2.5 Fixed Threshold Mode

In fixed threshold mode, you can specify the minimum amplitude level to detect. Any object reflections with an amplitude below the minimum level, will be ignored by the detector. To configure the Distance Peak Detector to operate in fixed threshold mode, call the acc_detector_distance_peak_set_threshold_mode_fixed function.

```
acc_detector_distance_peak_status_t detector_status;

acc_detector_distance_peak_configuration_t distance_configuration;

distance_configuration = acc_detector_distance_peak_configuration_create();

detector_status = acc_distance_set_detector_threshold_mode_fixed(
    distance_configuration, FIXED_THRESHOLD_VALUE);
```

2.6 Stationary Clutter Threshold Mode

The client can choose to use a fixed threshold as described above or use the "stationary clutter estimated threshold". In "stationary clutter estimated threshold" mode, first the detector records background reflections from stationary objects in the environment surrounding the sensor. A threshold varying with distance is then calculated, so that the amplitude of the reflections from the stationary objects will be located below the threshold level at the distances where the objects are located. At distances with no stationary clutter, the threshold level will be lower. To set up a detector in this mode call the acc_detector_distance_peak_threshold_estimation_update function.

You are recommended to use at least 50 updates with background reflections containing stationary clutter before using the Distance Peak Detector.

A new threshold estimation should be performed if significant changes were made in the sensor's surrounding environment. To reset the Distance Peak Detector to empty state, please call the acc_detector_distance_peak_threshold_estimation_reset function. Then update the Distance Peak Detector for the new environment using the acc_detector_distance_peak_threshold_estimation_update function.

```
acc_detector_distance_peak_status_t detector_status;

detector_status = acc_detector_distance_peak_threshold_estimation_update(
   distance_configuration, 100, metadata.start_m, metadata.start_m +
   metadata.length_m);
```

It is possible to control the sensitivity and false detection rate of the Distance Peak Detector in estimated threshold mode. With high sensitivity, the detector is more likely to make false detections, e.g. interpret noise as an object. At the same time, the number of missed detections is low. With low sensitivity, the number of missed detections is likely to increase, whereas false detections are likely to decrease.

The sensitivity of the detector is set when calling the acc_detector_distance_peak_set_sensitivity function. This function takes a sensitivity parameter in the range between 0 and 1, where the 0 represents the lowest sensitivity and 1 the highest. The function is optional but must be called before before activating the detector.

3 Measure Distances

3.1 Creating and Activating the Distance Peak Detector

After the detector configuration has been prepared and populated with desired configuration parameters, the actual Distance Peak 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_distance_peak_handle_t handle =
   acc_detector_distance_peak_create(distance_configuration);
```

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

```
acc_detector_distance_peak_metadata_t   metadata;

acc_detector_distance_peak_get_metadata(handle, &metadata);

float start_m = metadata.start_m;
float end_m = metadata.start_m + metadata.length_m;
```

To activate the detector call the acc_detector_distance_peak_activate function. Now the detector is producing detector data which might be retrieved by calling the acc_detector_distance_peak_get_next function.

```
detector_status = acc_detector_distance_peak_activate(handle);

if (detector_status != ACC_DETECTOR_DISTANCE_PEAK_STATUS_SUCCESS) {
/* Handle error */
}
```

3.2 Getting Detection Results

When the detector has been created and activated the detections results may be retrieved by calling the acc_detector_distance_peak_get_next function.

```
acc_detector_distance_peak_configuration_t distance_configuration;
acc_detector_distance_peak_handle_t        handle;
acc_detector_distance_peak_status_t        detector_status;
uint_fast16_t                              reflection_count = 10;
acc_detector_distance_peak_reflection_t    reflections[reflection_count];

distance_configuration = acc_detector_distance_peak_configuration_create();

handle = acc_detector_distance_peak_create(distance_configuration);

detector_status = acc_detector_distance_peak_activate(handle);
```

```
detector_status = acc_detector_distance_peak_get_next(handle, reflections, &
    reflection_count, &result_info);
```

To get the actual distances, we must start by allocating memory for an array of type acc_detector_distance_peak_reflection_t, for storing distance estimations. In the example above, this array is allocated on the stack. Then we can call acc_detector_distance_peak_get_next to fill the array with distances and amplitudes for such reflections, which have been detected as objects by the Distance Peak Detector.

4 Deactivating and Destroying the Distance Peak Detector

To release the memory resources allocated by the Distance Peak Detector, please call the acc_detector_distance_peak_deactivate function followed by the acc_detector_distance_peak_destroy function and finally by calling the acc_detector_distance_peak_configuration_destroy function. Do this when you have reached the point where you do not need to use the detector anymore.

```
detector_status = acc_detector_distance_peak_deactivate(handle);
acc_detector_distance_peak_destroy(&handle);
acc_detector_distance_peak_configuration_destroy(&distance_configuration);
```

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.