Envelope Service

User Guide

Envelope Service

User Guide

Author: Acconeer AB

Version:v2.0.0

Acconeer AB December 2, 2019

Contents

1 Envelope Service

The Envelope Service is one of four services that provides an interface for reading out the radar signal from the Acconeer A111 sensor. The data returned from the Envelope service is typically further processed and can be used in different types of algorithms such as distance measurement algorithms, motion detection algorithms, and object positioning algorithms. In use cases where low computation complexity is important, and the exact location of objects is less important you may consider using the power bins service instead of the Envelope service. Advanced users that needs phase information for detecting very small variations in distance will probably prefer the IQ data service instead of the envelop service. Users which are interested in detection of any movement, small or large, occurring in front of the sensor can instead use our sparse service.

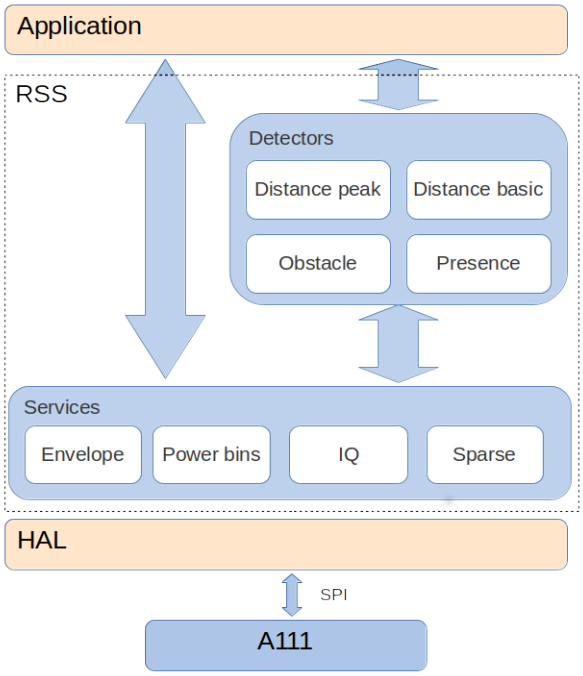

Figure 1:

Acconeer also provide several easy to use detectors that are implemented on top of the basic data services. The detectors provide an interface for higher level tasks like distance measurements, motion detection etc.

Acconeer provides an example of how to use the Envelope service: example_service_envelope.c

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.

1.1 Disclaimer

Profile 3-5 will not have optimal performance using A111 with batch number 10467, 10457 or 10178 (also when mounted on XR111 and XR112). XM112 and XM122 are not affected since they have A111 from other batches.

2 Setting up the Service

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 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 */
}
```

2.2 Service API

All services 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 service, 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 service is activated the radar is activated and can start producing data.

2.3 Envelope Service Configuration

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

```
acc_service_configuration_t envelope_configuration =
    acc_service_envelope_configuration_create();

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

The newly created service configuration contains default settings for all configuration parameters and can be passed directly to the acc_service_create function. However, in most scenarios there is a need to change at least some of the configuration parameters. See acc_service_envelope.h and acc_base_configuration.h for a complete description of configuration parameters.

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 wavelet length and optimize on either depth resolution or radar loop gain. More information regarding profiles can be read in the sensor introduction document, https://acconeer-python-exploration.readthedocs.io/en/latest/sensor_introduction.html.

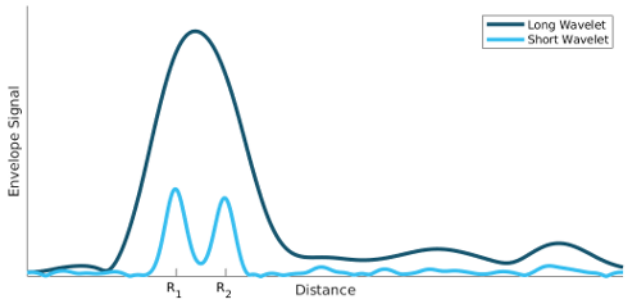

Figure 2:

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

The Envelope service 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 wavelets 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 service api. The default profile is ACC_SERVICE_PROFILE_2.

```
void acc_service_profile_set(acc_service_configuration_t
    service_configuration,
                             acc_service_profile_t       profile);
```

2.3.2 Repetition Mode

RSS supports two different repetition modes which configure the control flow of the sensor when it's producing data. In both modes, the application initiates the data transfer from the sensor and is responsible to keep the timing by fetching data from the service. The repetition modes are called on_demand and streaming and the default mode is on_demand.

Repetition mode on_demand lets the application decide when the sensor produces data. This mode is recommended to be used if the application is not dependent of a fixed update rate and it's more important for the application to control the timing. An example could be if the application requests data at irregular time or with low frequency and it's more important to enable low power consumption. Repetition mode on_demand should also be used if the application set a length which requires stitching or want to use power save mode off.

```
void acc_base_configuration_repetition_mode_on_demand_set(
    acc_base_configuration_t configuration);
```

Repetition mode streaming configures the sensor to produce data based on a hardware timer which is very accurate. It is recommended to use repetition mode streaming if the application requires very accurate timing. An example could be if the data should be processed with a fft. This mode can not be used it the application set a length which requires stitching.

```
void acc_base_configuration_repetition_mode_streaming_set(
    acc_base_configuration_t configuration, float update_rate);
```

2.4 Creating Service

After the Envelope configuration has been prepared and populated with desired configuration parameters, the actual Envelope service 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 service and get data from the sensor (unless there is some unexpected hardware error).

```
acc_service_handle_t handle = acc_service_create(envelope_configuration);

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

During service create, the service run a calibration sequence on the sensor. The calibration is used once at create and can be used until the service is destroyed. A new calibration is needed if the environment is changed, such as deviation in temperature.

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

When the service has been created it is possible to get the actual number of samples (data_length) we will get for each result. This value can be useful when allocating buffers for storing the Envelope data.

```
acc_service_envelope_metadata_t envelope_metadata;
acc_service_envelope_get_metadata(handle, &envelope_metadata);

uint16_t data[envelope_metadata.data_length];
```

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

```
if (!acc_service_activate(handle))
{
    /* Handle error */
}
```

2.5 Reading Envelope Data from the Sensor

Envelope data is read from the sensor by a call to the function acc_service_envelope_get_next. This function blocks until the next sweep arrives from the sensor and the result is available in envelope_data.

```
uint16_t                          data[envelope_metadata.data_length];
acc_service_envelope_result_info_t result_info;

for (int i = 0; i < 10; i++)
{
    if (!acc_service_envelope_get_next(handle, data, envelope_metadata.
        data_length, &result_info))
    {
        /* Handle error */
    }
}
```

2.6 Deactivating and Destroying the Service

Call the acc_service_deactivate function to stop measurements.

```
if (!acc_service_deactivate(handle))
{
    /* Handle error */
}
```

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

```
acc_service_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();
```

3 How to Interpret the Envelope Data

The Envelope data should be interpreted as a single one-dimensional array, where each array element represents the amplitude of the reflected signal from objects at a particular radial distance from the sensor.

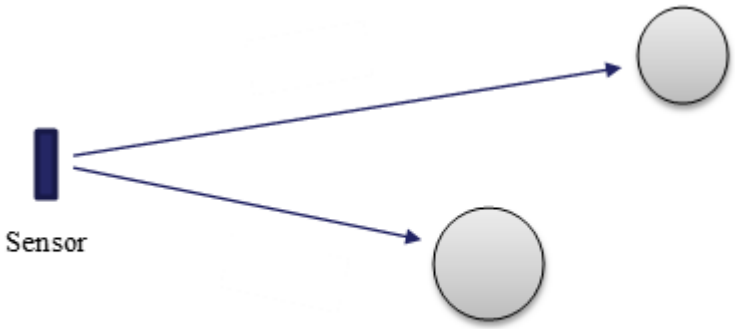

Figure 2- Measuring two objects

Figure 3:

Assume that you have a setup with two objects like in the figure above. When plotting the Envelope data recorded using profile 1, which is optimized for maximum depth resolution, you will get a graph similar to the one below.

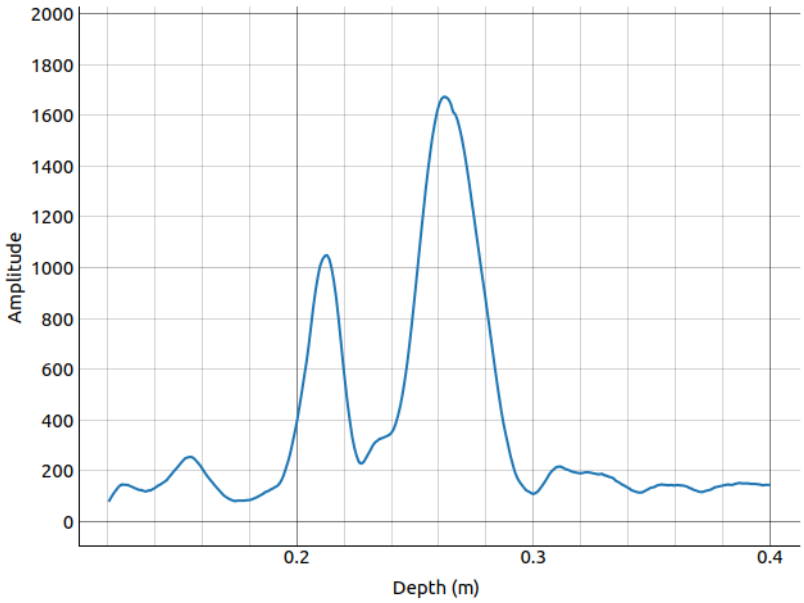

Figure 4:

When a profile is choosen to improve radar loop gain, the radar sends out more energy, and the receiver has a higher gain setting which leads to a higher amplitude in the received signal. The internal filter parameters are also different which gives a smoother signal where noise and fine details have been removed. The graph below shows the same two objects as before but with profile 2 enabled.

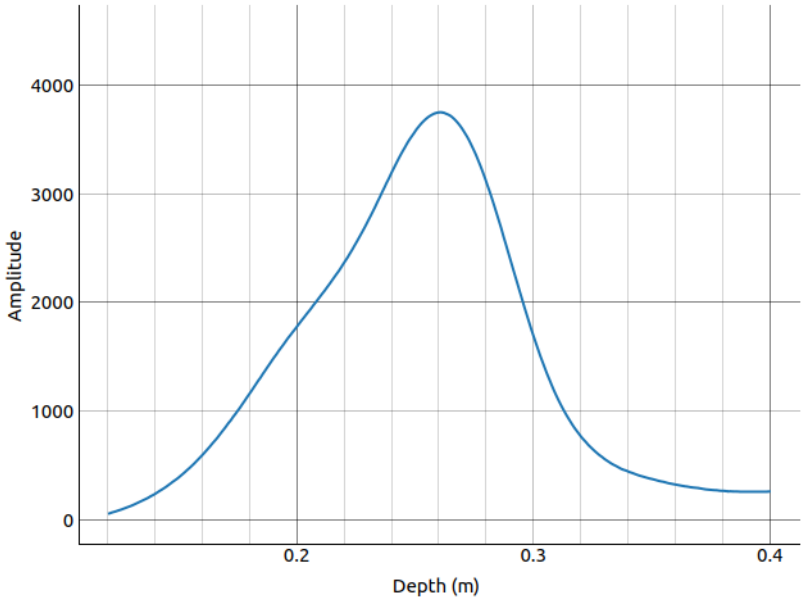

Figure 5:

3.1 Envelope Metadata

In addition to the array with Envelope data samples, a metadata data structure provides side information that can be useful when interpreting the Envelope data. This metadata can be retrieved after creating the service. It will not change during operation, so it is only needed to be retrieved once for the created service.

```
acc_service_envelope_metadata_t envelope_metadata;
acc_service_envelope_get_metadata(handle, &envelope_metadata);
```

The most important member variable in the meta data structure is data_length which holds the length of the Envelope data array. For other member variables see acc_service_envelope_metadata_t.

3.2 Envelope Result Info

Result info is another kind of metadata which might change for each retrieved result. Result info is provided at the same time as the resulting array, either when calling get_next() or when a callback is triggered.

```
acc_service_envelope_result_info_t result_info;
acc_service_envelope_get_next(handle, data, data_length, &result_info);
```

The most important member variable is the sensor_communication_error which indicates whether a sensor communication has occurred. For other member variables see acc_service_envelope_result_info_t

4 Examples

4.1 Simple Distance Algorithm Example

In this example we will implement a simple distance algorithm that finds the distance to the strongest reflecting object by finding the highest peak in the Envelope data. We start by writing a function to find the index for the value in the Envelope data array with the highest amplitude.

```
int peak_index(uint16_t data[], uint16_t data_length)
{
    uint16_t max = 0;
    int peak_idx = 0;
    for (int i = 0 ; i<data_length ; i++)
    {
           if (data[i] > max )
```

```
        {
            max = data[i];
            peak_idx = i;
        }
    }
    return peak_idx;
}
```

By using information on start and length in the meta data structure we can transform the index to a distance. To avoid measuring the distance to some peak in the background noise when there is no object in front of the sensor we define an amplitude threshold that must be exceeded in order to write out that we found a distance.

```
int peak_idx = peak_index(envelope_data, envelope_metadata.data_length);
unint16_t min_amplitude = 1000;

if (envelope_data[peak_idx] > min_amplitude)
{
   float dist =  envelope_metadata.start_m + peak_idx *
                 envelope_metadata.step_length_m;
   printf ("distance: %f, amplitude %u\n", dist, envelope_data[peak_idx]);
}
else
{
   printf ("amplitude under threshold\n");
}
```

4.2 Direct Leakage Measurement

In this example we will outline how to set up the service to do a direct leakage measurement. Note that only the configuration of the service is shown. Refer to previous chapters on how to create and activate the service and how to read out the service data.

```
acc_service_configuration_t envelope_configuration =
   acc_service_envelope_configuration_create();

if (envelope_configuration == NULL)
{
    /* Handle error */
}

acc_service_maximize_signal_attenuation_set(envelope_configuration, true);
   /* This is the main switch for performing direct leakage measurements */

sweep_configuration = acc_service_get_sweep_configuration(
   envelope_configuration);
if (sweep_configuration == NULL)
{
    /* Handle error */
}

acc_sweep_configuration_hw_accelerated_average_samples_set(
   sweep_configuration, 1); /* Sampling each data point one time is suitable
    for this measurement */
acc_sweep_configuration_receiver_gain_set(sweep_configuration, 0.09);
                 /* Lower the receiver gain to not saturate the signal */
acc_sweep_configuration_requested_range_set(sweep_configuration, -0.12,
   0.12);      /* Set the range where the direct leakage is visible */

/* Create, activate and read out the service data */
```

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.