Envelope Service

User Guide

Envelope Service

User Guide

Author: Acconeer AB

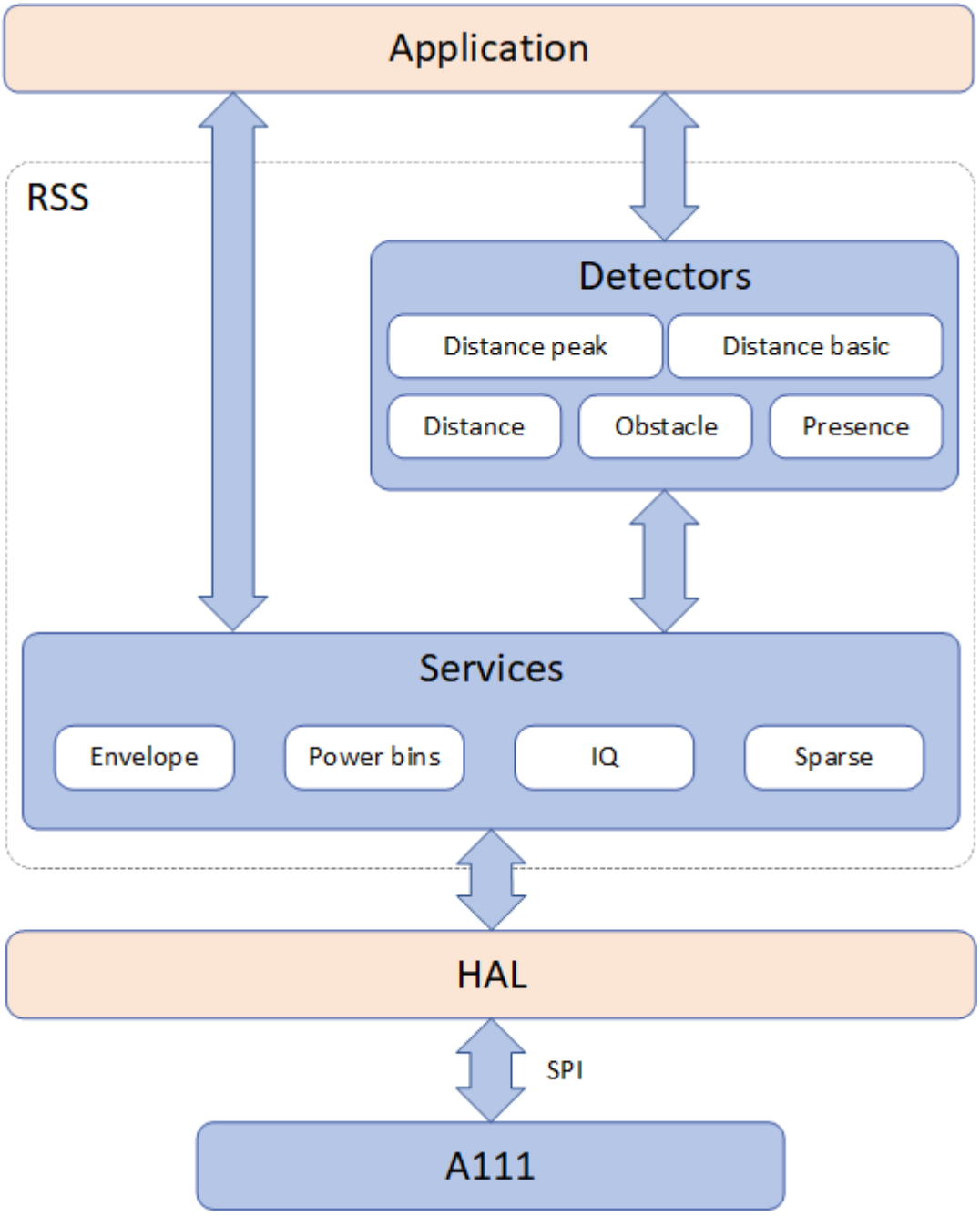
Version:a111-v2.15.2

Acconeer AB August 18, 2023

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 envelope 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.



Acconeer also provides 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 on 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 Acconeer Exploration Tool.

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.

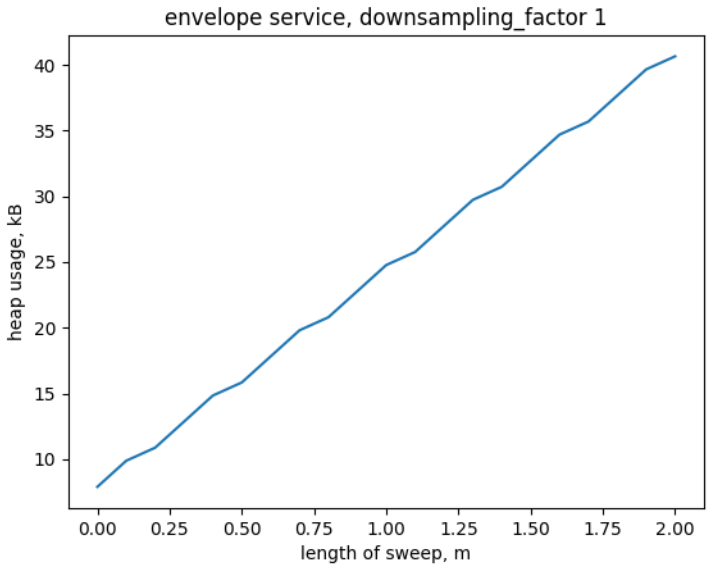
2 Memory Requirements

2.1 Flash footprint

The different services in RSS have different flash footprint depending on the complexity of the algorithm used to process the data. The example program for the Envelope service requires 90 kB flash when built in STM32Cube environment.

2.2 Heap usage

The amount of heap used by the Envelope service depends on how the application configures the service. Different sweep length and downsampling factor impact the size of the buffer needed to keep the service data. A downsampling factor of 1 gives better accuracy in the measurement but a downsampling factor of 4 can be used in applications where less heap usage is more important than accuracy. The client can also choose to use acc_service_envelope_get_next() if it wants to save the envelope data in a separate buffer or acc_service_envelope_get_next_by_reference() if it wants to use the same buffer as is used in the service.



The figure above gives an indication of how much memory that is allocated on the heap in relation to sweep length and acc_service_envelope_get_next() is used to retrieve the data.

3 Setting up the Service

3.1 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, 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 */
}
```

3.2 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.

3.3 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.

3.4 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_service.h for a complete description of configuration parameters.

3.4.1 Configuration Summary

Below is two tables of all possible configurations for the Envelope Service. Note that some configurations can have other limits than the ones listed below. For example, 'Length' in combination with 'Downsampling Factor' is dependent on the available memory of the system.

Generic Configurations :

Parameter	Description	Type	Unit	Limits
Sensor	Sensor ID	integer	N/A	[1 -]
Start	Start of measurement	float	meters	[-0.7 - 7.0]
Length	Length of measurement	float	meters	[0.0 - 7.7]
Repetition Mode	See below	On demand / Streaming	N/A	N/A
Power Save Mode	See below	enum	N/A	N/A
Receiver Gain	Sensor receiver gain	float	N/A	[0.0 - 1.0]
TX Disable	Disable Radio Transmitter	bool	N/A	N/A
HWAAS	See below	integer	N/A	[1 - 63]
Profile	See below	enum	N/A	N/A
Maximize Signal Attenuation	Maximize signal attenuation in sensor	bool	N/A	N/A
Asynchronous Measurement	Enable Asynchronous Measurement	bool	N/A	N/A

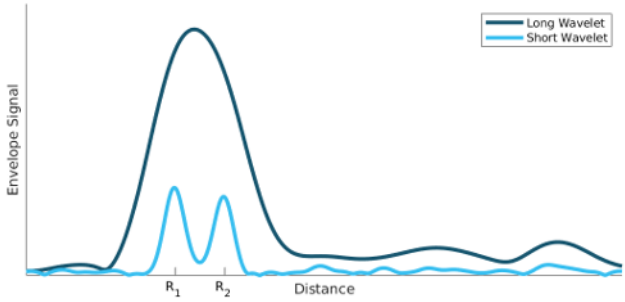
Envelope Specific Configurations :

Parameter	Description	Type	Unit	Limits
Downsampling Factor	See below	integer	N/A	[1, 2, 4]
Running Average Factor	Factor to determine how much the data is time filtered	float	N/A	[0.0 - 1.0]
Noise Level Normalization	Flag to determine if data should be normalized with noise floor	bool	N/A	N/A

The following sections describe some of the configurations in more detail.

3.4.2 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 Envelope service supports five 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 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);
```

3.4.3 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_service_repetition_mode_on_demand_set(acc_service_configuration_t
   service_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 an FFT. This mode can not be used if the application has set a length which requires stitching.

```
void acc_service_repetition_mode_streaming_set(acc_service_configuration_t
    service_configuration, float update_rate);
```

3.4.4 Downsampling Factor

In the Envelope service, 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 on 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 consumption. The Envelope service supports a downsampling factor of 1, 2, or 4.

```
void acc_service_envelope_downsampling_factor_set(
    acc_service_configuration_t service_configuration, uint16_t
    downsampling_factor);
```

The actual step length is reported back from the service in acc_service_envelope_metadata_t.

3.4.5 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 Envelope service is 10.

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

3.4.6 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, and hardware accelerated average samples. 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_service_power_save_mode_set(acc_service_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.
- Fetching a burst of frames: Some applications need to fetch a burst of frames from the sensor and then sleep for a longer period. This kind of application is recommended to use mode ‘READY’ for fast update rate between the frames in the burst to minimize the execution time when the MCU is active. To save maximum power it is recommended to deactivate the service between the bursts. This will put the sensor in power-off state when it is not used.

3.4.7 Asynchronous Measurement

RSS supports two different measurement modes, synchronous and asynchronous. The default mode is asynchronous.

In synchronous mode, the following will occur when acc_service_envelope_get_next() / acc_service_envelope_get_next_by_reference() is called:

1. Start the sweep.
2. Wait for the sweep to finish, the time for this will vary depending on the configuration and the sweep length.
3. Transfer sweep data from the sensor.
4. Process data.
5. Return from function.

In asynchronous mode, the following will occur when acc_service_envelope_get_next() / acc_service_envelope_get_next_by_reference() is called:

1. Wait for previous sweep to finish
2. Transfer sweep data from the sensor.
3. Start the next sweep.
4. Process data.
5. Return from function.

The main difference between the modes is that in asynchronous mode the host can do work while the sensor is finishing the sweep. Since the sensor and the host can do work in parallel the update rate of the system will be higher in asynchronous mode. In asynchronous mode the call to acc_service_envelope_get_next() / acc_service_envelope_get_next_by_reference() will actually acquire the data from the sweep that was started by the previous call to acc_service_envelope_get_next() / acc_service_envelope_get_next_by_reference().

In synchrounous mode the sensor is garanteed to be idle outside of the acc_service_envelope_get_next() / acc_service_envelope_get_next_by_reference() calls.

The synchronous mode in combination with streaming Repetition Mode will result in a failure when trying to create the service.

```
void acc_service_asynchronous_measurement_set(acc_service_configuration_t
    configuration, bool asynchronous_measurement);
```

3.4.8 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_service_mur_set(acc_service_configuration_t service_configuration,
                         acc_service_mur_t           max_unambiguous_range);
```

3.5 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 */
}
```

3.6 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 data. When using this function, it is up to the application to allocate memory for the result, as can be seen below.

```
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 */
    }
}
```

Another way to get the data is to call the function acc_service_envelope_get_next_by_reference. This function also blocks until the next sweep arrives from the sensor and the result is available in data. The difference from the function above is that RSS will provide the memory in the resulting data. The length of the data is still provided in envelope_metadata.data_length. The memory provided is owned by RSS and should not be freed. The application is however free to manipulate the data until the next call to acc_service_envelope_get_next_by_reference. The reason to use this function is the reduced ram usage for the application as well as increased speed for the function call.

```
uint16_t                            *data;
acc_service_envelope_result_info_t  result_info;

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

3.7 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.

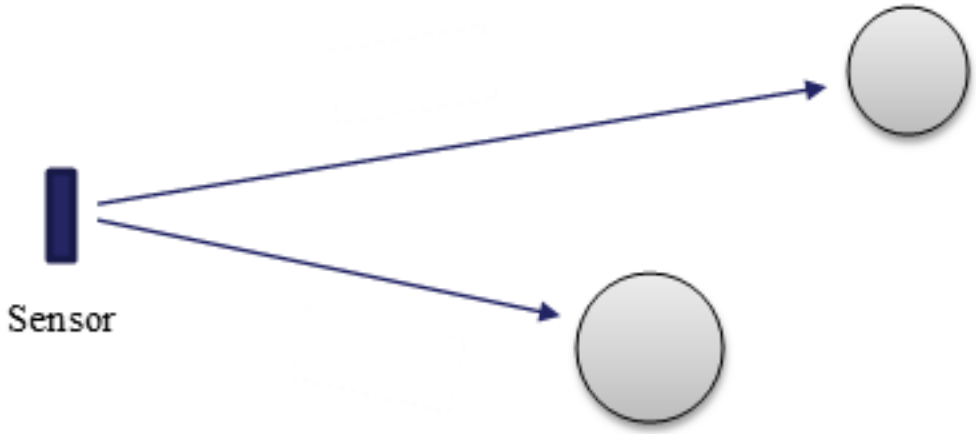
```
acc_service_destroy(&handle);
```

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.

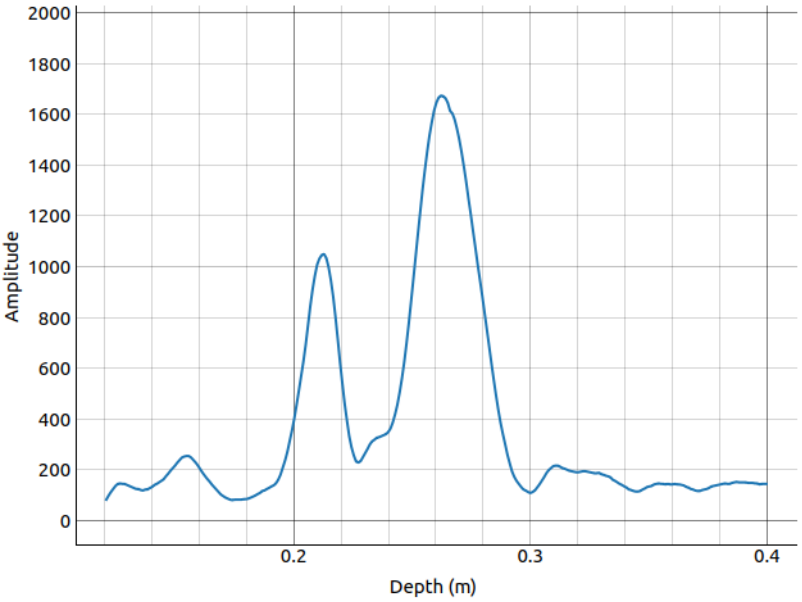
```
acc_rss_deactivate();
```

4 How to Interpret the Envelope Data

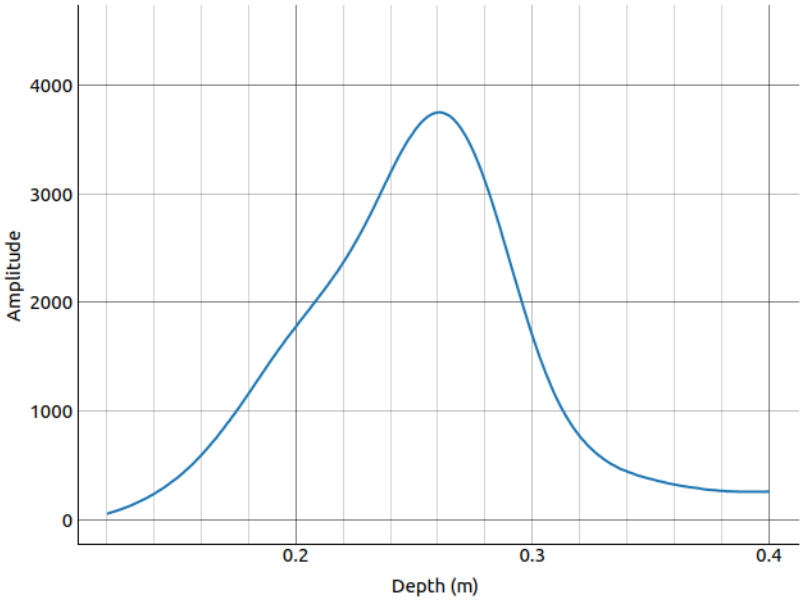
The Envelope data should be interpreted as a single one-dimensional array, where each array element represents the amplitude of the reflected signal from objects at a particular radial distance from the sensor.



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.



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.



4.1 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.

4.2 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 member variables sensor_communication_error and data_quality_warning are intended for continuous monitoring from the application. A true value of sensor_communication_error indicates a hardware-related failure to obtain data from the sensor. The sensor can end up in a state that the service does not recover from. Therefore, it's recommended to destroy the service and create it again if there is a communication error. A true value of data_quality_warning indicates that an internal sensor parameter is outside its interval for normal operation. This issue is likely to occur when the temperature of the sensor has changed and the sensor needs a new calibration. The service performs a calibration when it is created and it is recommended to destroy the service and create it again when an application receive a data_quality_warning.

For other member variables see acc_service_envelope_result_info_t.

5 Examples

5.1 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");
}
```

5.2 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 */
acc_service_hw_accelerated_average_samples_set(envelope_configuration, 1);
   /* Sampling each data point one time is suitable for this measurement */
acc_service_receiver_gain_set(envelope_configuration, 0.09);
   /* Lower the receiver gain to not saturate the signal */
acc_service_requested_start_set(envelope_configuration, -0.12);
   /* Set the start where the direct leakage is visible */
acc_service_requested_length_set(envelope_configuration, 0.24);
   /* Set the length where the direct leakage is visible */

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

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.