Power Bins Service

User Guide

Power Bins Service

User Guide

Author: Acconeer AB

Version:v2.0.0

Acconeer AB December 2, 2019

Contents

1 Power Bins Service **3**
1.1 Disclaimer 3

2 Setting up the Service **3**
2.1 Initializing the System 3
2.2 Service API 4
2.3 Power Bins Service Configuration 4
2.3.1 Profiles 4
2.3.2 Repetition Mode 5
2.4 Creating Service 5
2.5 Reading Power Bins Data from the Sensor 6
2.6 Deactivating and Destroying the Service 6

3 How to Interpret the Power Bins Data **7**
3.1 Power Bins Metadata 7
3.2 Power Bins Result Info 7

4 Object Detection Example **8**

5 Disclaimer **11**

1 Power Bins Service

The Power Bins Service is one of four services that provide an interface for reading out the radar signal from the Acconeer A111 sensor. The Power Bins Service is mainly intended for use in low power applications where large objects are measured at short distances from the radar sensor e.g. in parking sensors mounted on the ground.

The data provided by the Power Bins API is essentially subsampled envelope data. However, the Power Bins values are calculated using an algorithm optimized for low computational complexity. This algorithm will not remove as much noise from the signal as the more advanced signal processing algorithms used in the Envelope and IQ Data Services.

For use cases where weaker radar echoes are measured or when the resolution of the signal is important, we recommend using the Envelope or IQ data service instead.

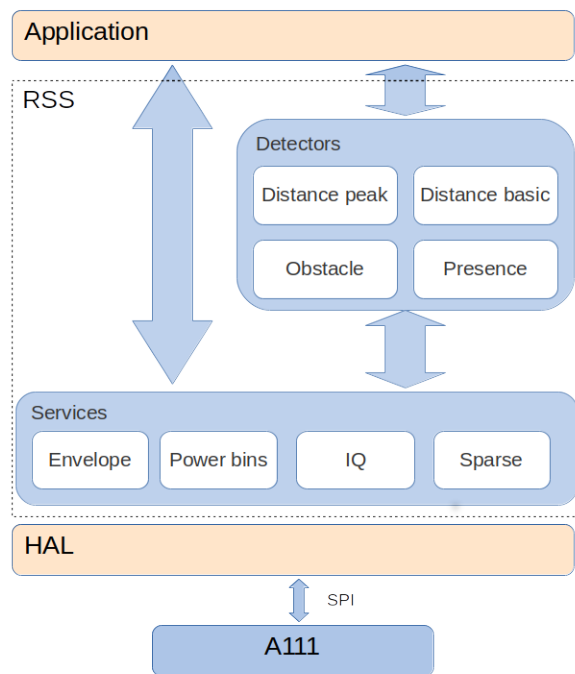

Figure 1:

Acconeer provide an example of how to use the Power Bins service: example_service_power_bins.c

For more details on the Power Bins 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 to the radar sensor service API. 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 */
}

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 the radar data starts to flow from the sensor to the application.

2.3 Power Bins Service Configuration

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

```
acc_service_configuration_t power_bins_configuration =
    acc_service_power_bins_configuration_create();

if (power_bins_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_power_bins.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.

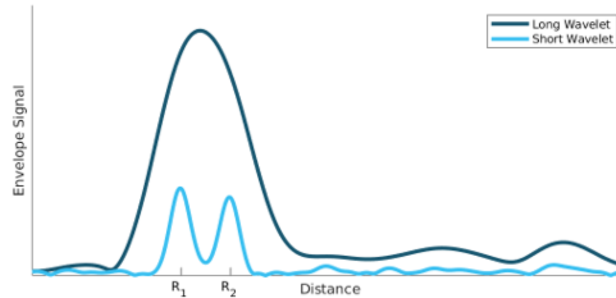

Figure 2:

The figure above use the Envelope signal to visualize the same objects with two different profiles, one with short wavelet and one with longer.

The Power Bins 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 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 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 Power Bins configuration has been prepared and populated with desired configuration parameters, the actual Power Bins 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(power_bins_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 sweep. This value can be useful when allocating buffers for storing the Power Bins data.

```
acc_service_power_bins_metadata_t power_bins_metadata;
acc_service_power_bins_get_metadata(handle, &power_bins_metadata);

uint16_t data[power_bins_metadata.data_length];
```

It is now also possible to activate the service. This means that the radar sensor starts to do measurements.

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

2.5 Reading Power Bins Data from the Sensor

Power Bins data is read from the sensor by a call to the function acc_service_power_bins_get_next. This function blocks until the next sweep arrives from the sensor and the Power Bins data is then copied to the power_bins_data array.

```
uint16_t                           data[power_bins_metadata.data_length];
acc_service_power_bins_result_info_t result_info;

for (int i = 0; i < 10; i++)
{
    if (!acc_service_power_bins_get_next(handle, data, power_bins_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 Power Bins Data

Each data point in a Power Bins data sweep contains the average amplitude in a short interval. In the example below, we are measuring the interval 0.15 m to 0.50 m using 9 bins. The interval is divided into 9 sub intervals and for each subinterval the average amplitude is calculated. The first subinterval (bin 1) contains the average amplitude for the interval 0.15 m to approximately 0.19 m and the last subinterval (bin 9) contains the average amplitude for the interval from approximately 0.46m to 0.5 m. The object is placed at roughly 24 cm and gives the highest amplitude in bin 3.

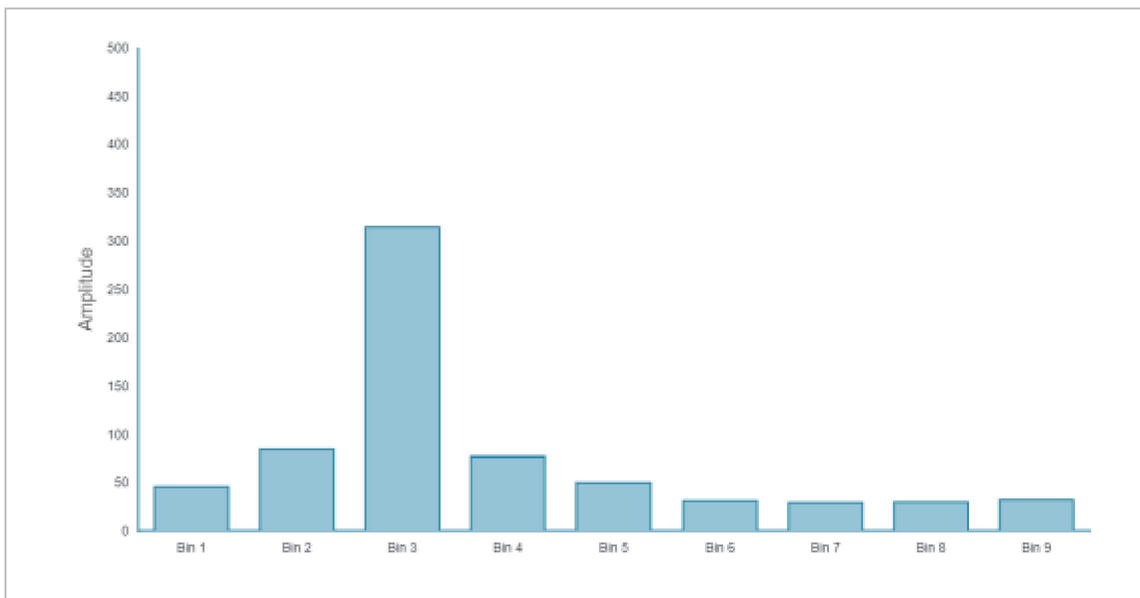

Figure 3:

3.1 Power Bins Metadata

In addition to the array with Power Bins data samples, a metadata data structure provides side information that can be useful when interpreting the Power Bins 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_power_bins_metadata_t power_bins_metadata;
acc_service_power_bins_get_metadata(handle, &power_bins_metadata);
```

The most important member variable in the meta data struct is bin_count. It contains the number of entries in the Power Bins data array. For other member variables see acc_service_power_bins_metadata_t.

3.2 Power Bins 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_power_bins_result_info_t result_info;
acc_service_power_bins_get_next(handle, data, data_length, &result_info);
```

The most important member variable is the sensor_communication_error which indicates whether an error occurred when communicating with the sensor. For other member variables see acc_service_power_bins_result_info_t

4 Object Detection Example

In the following example we will show how the Power Bins API can be used for detecting if an object is present in front of the sensor. The basic idea is to detect if the signal amplitude in any of the Power Bin data values are exceeding a predefined threshold. The measured amplitude can differ between different sensor samples. This can be problematic in applications like this where we compare the amplitude to a predefined threshold. To reduce the impact of the problem we first record the noise level. We then use the average noise level to normalize the measured values when transmission is enabled. By using this trick, we can compensate for gain variations in the receiver chain of the radar sensor.

First code example show how to measure the noise level for the sensor.

```
acc_service_configuration_t configuration;
acc_service_handle_t handle;

configuration = acc_service_power_bins_configuration_create();

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

acc_service_power_bins_requested_bin_count_set(configuration, 10);

acc_base_configuration_t base_configuration;

base_configuration = acc_service_get_base_configuration(configuration);

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

// Update base configuration according to project needs

// Disable radio transmitter to only capture noise
acc_base_configuration_tx_disable_set(base_configuration, true);

handle = acc_service_create(configuration);

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

acc_service_power_bins_configuration_destroy(&configuration);

handle = setup_power_bins(disable_transmission);
acc_service_power_bins_get_metadata(handle, &power_bins_metadata);

float  power_bins_data[power_bins_metadata.bin_count];
float sum = 0;
const int n = 10;

bool status = acc_service_activate(handle);

for (int i=0 ; i<n ; i++) {
    status = acc_service_power_bins_get_next(handle,
                                             power_bins_data,
                                             power_bins_metadata.bin_count,
                                             &result_info);

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

```
    for (int j=0 ; j < power_bins_metadata.bin_count ; j++) {
        sum += power_bins_data[j];
    }

}

status = acc_service_deactivate(handle);

if (!status) {
    /* Handle error */
}

acc_service_destroy(&handle);

float noise_level = sum / (n * power_bins_metadata.bin_count);
```

The second part detect if an object is present in front of the sensor. It uses the noise level from previous code example as a normalization factor

```
acc_service_configuration_t configuration;
acc_service_handle_t handle;
bool object_present = false;

configuration = acc_service_power_bins_configuration_create();

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

acc_service_power_bins_requested_bin_count_set(configuration, 10);

acc_base_configuration_t base_configuration;

base_configuration = acc_service_get_base_configuration(configuration);

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

// Update base configuration according to project needs

handle = acc_service_create(configuration);

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

acc_service_power_bins_configuration_destroy(&configuration);

handle = setup_power_bins(disable_transmission);
acc_service_power_bins_get_metadata(handle, &power_bins_metadata);

float  power_bins_data[power_bins_metadata.bin_count];
float sum = 0;
const int n = 10;

bool status = acc_service_activate(handle);

for (int i=0 ; i<n ; i++) {
    status = acc_service_power_bins_get_next(handle,
```

```
                                                      power_bins_data,
                                                      power_bins_metadata.bin_count,
                                                      &result_info);

    if (!status) {
        /* Handle error */
    }

    for (int j=0 ; j < power_bins_metadata.bin_count ; j++) {
        if (radar_data[i] / noise_level > threshold){
            object_present =  true;
        }
    }

}

status = acc_service_deactivate(handle);

if (!status) {
    /* Handle error */
}

acc_service_destroy(&handle);
```

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.