Distance Detector

User Guide

A111 – Distance Detector

User Guide

Author: Acconeer

Version 1.0: 2018-02-12

Acconeer AB

Table of Contents

1. Using the Distance Detector

The distance detector can operate in two different modes, a “fixed threshold mode”, and a “stationary clutter threshold mode”.

1.1 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 create a distance detector operating in fixed threshold mode, call the **acc_detector_distance_create_fixed** function.

```
void *distance_detector;

distance_detector = acc_detector_distance_create_fixed(FIXED_THRESHOLD_VALUE);
if (distance_detector == NULL) {
        /* Handle error */
}
```

When the distance detector has been created, it can be used for distance measurements. As a first step, envelope data is sent to the distance detector by calling the **acc_detector_distance_detect** function. This function has a **reflection_count** out parameter, used for obtaining the actual distances in a second step.

```
uint16_t reflection_count;

status = acc_envelope(sensor, range_start, range_end, envelope_result);
if (status != ACC_STATUS_SUCCESS) {
              /* Hande error */
}

status = acc_detector_distance_detect(distance_detector, envelope_result.actual_start,
               envelope_result.actual_end, envelope_result.data_size,
               envelope_result.data, &reflection_count);
if (status != ACC_STATUS_SUCCESS) {
              /* Hande error */
}
```

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

```
acc_detector_distance_reflection_t reflections[reflection_count];

if (reflection_count > 0) {

              status = acc_detector_distance_get_reflections(distance_detector, §
                           reflection_count, reflections);
              if (status != ACC_STATUS_SUCCESS) {
                           /* Handle error */
              }
}
```

To release the memory resources allocated by the distance detector, please call the **acc_detector_distance_destroy** function. Do this when you have reached the point where you do not need to use the detector anymore.


```
acc_detector_distance_destroy(distance_detector);
```

1.2 Stationary Clutter Threshold Mode

In "stationary clutter estimated threshold" mode, first you record 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 distance detector in this mode, first create a non-initialized distance detector using the **acc_detector_distance_create_empty** function.

```
void *distance_detector;

distance_detector = acc_detector_distance_create_empty();
if (distance_detector == NULL) {
              /* Handle eroor */
}
```

After creation of the empty distance detector, it must be trained for the current environment. This is done by sending envelope data to the distance detector by calling the **acc_detector_distance_threshold_estimation_update** function repeatedly. You are recommended to perform at least 50 updates with background reflections containing stationary clutter before using the distance detector.

A new threshold estimation should be performed if significant changes were made in the sensor's surrounding environment. To reset the distance detector to empty state, please call the **acc_detector_distance_threshold_estimation_reset** function. Then update the distance detector with envelope data from the new environment using the **acc_detector_distance_threshold_estimation_update** function.

It is possible to control the sensitivity and false detection rate of the distance 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_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 the first call to **acc_detector_distance_detect** if used.

```
acc_service_configuration_t envelope_configuration;
acc_service_handle_t envelope_handle;

acc_envelope_t             envelope_result;
acc_status_t               status;
acc_service_status_t       service_status

float                      sensitivity = 0.26;


envelope_configuration = acc_service_envelope_configuration_create();

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

/* configure envelope serveice */
envelope_handle = acc_service_create(envelope_configuration);


status = acc_detector_distance_threshold_estimation_reset(distance_detector);
if (status != ACC_STATUS_SUCCESS) {
   /* Handle error */
}

status = acc_detector_distance_set_sensitivity(distance_detector, sensitivity);
if (status != ACC_STATUS_SUCCESS) {
   /* Handle error */
}

uint16_t envelope_data[envelope_metadata->data_length];
float actual_start = envelope_metadata->actual_start_m;
float actual_end = envelope_metadata->actual_start_m + envelope_metadata->actual_length_m;

for (int i=0 ; i<50 ; i++)
{
    service_status = acc_service_envelope_get_next(envelope_handle,
                                                   envelope_data,
                                                   envelope_metadata->data_length);

    if (service_status != ACC_SERVICE_STATUS_OK) {
        /* handle error */
    }

    status = acc_detector_distance_threshold_estimation_update(distance_detector,
                                          actual_start,
                                          actual_end,
                                          envelope_metadata->data_length,
                                          envelope_data);

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

1.3 Calculating Envelope Average

The range and accuracy of distance measurements can be improved when collecting multiple envelope sweeps, and feeding the distance detector with average envelope data. The code sample below shows how envelope data averaging can be implemented.

```
acc_status_t get_avg_envelope_data(struct acc_service_handle *envelope_handle,
                                   acc_service_envelope_metadata_t *envelope_metadata,
                                   uint16_t number_sweeps_to_average,
                                   uint16_t *envelope_data,
                                   uint16_t envelope_data_length)
{
    acc_status_t    status;
    uint32_t        data_sum[envelope_data_length] = {0};

    for (int sweep = 0 ; sweep < number_sweeps_to_average ; sweep++) {

        service_status = acc_service_envelope_get_next(envelope_handle,
                                                       envelope_data,
                                                       envelope_metadata->data_length);

        if (service_status != ACC_SERVICE_STATUS_OK) {
             /* handle error */
        }

        for (int i = 0 ; i < envelope_data_length ; i++) {
            data_sum[i] += envelope_result->data[i];
        }
    }

    for (int i = 0 ; i < envelope_data_length ; i++) {
        envelope_result->data[i] = data_sum[i] / number_sweeps_to_average;
    }

    return ACC_STATUS_SUCCESS;
}
```

2. Measuring Absolute Distances

There is a small offset error in distances returned by the distance sensor. This may be caused by multiple factors, such as the placement of the sensor in relation to the ground plane, materials covering the sensor and manufacturing process variations.

The sensor specific offset error can be reduced when subtracting the **free_space_absolute_offset**, returned as a result from the call to **acc_sensor_preparation_receive**. To compensate for other sources of offset error, related to the placement of the sensor and surrounding materials, the offset error can be estimated to:

offet_error = a * free_space_absolute_offset + b

The constants **a** and **b** are design specific and depend on electrical environmental factors, such as PCB layout and materials covering the sensor.

3. Additional Settings

As of release SW v1.1.28, the amplitude values returned by the distance detector constitute the difference between the reflection amplitude and the threshold. The **acc_detector_distance_set_absolute_amplitude** function can be called to configure the distance detector to legacy behavior and return absolute amplitude values.

```
acc_detector_distance_set_absolute_amplitude(distance_detector, true);
```

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.