Obstacle Detector

User Guide

Obstacle Detector

User Guide

Author: Acconeer AB

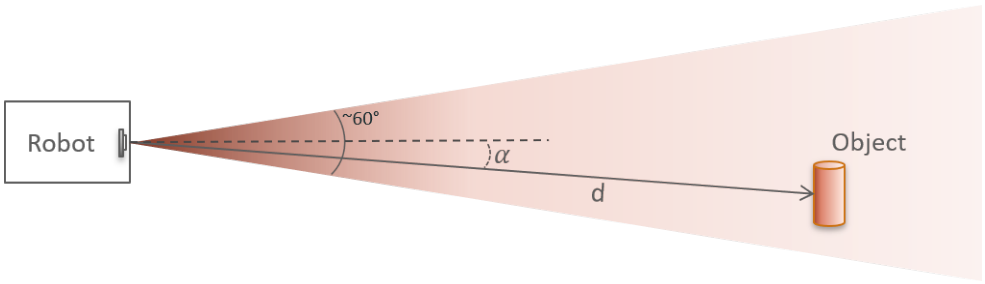
Version:a111-v2.15.2

Acconeer AB August 18, 2023

Contents

1 Obstacle Detector

The obstacle detector algorithm is based on the synthetic aperture radar (SAR) principle. The main goal of the detector is to find obstacles in front of the sensor and estimate their distance and angle. In an application where the sensor is installed on a robot (such as vacuum cleaners or lawn mowers), the object data can be used to issue movement commands to the robot to avoid and circumnavigate the found obstacles.



Due to a half power beam width (HPBW) of 60 degrees along the H-plane of the sensor, detection of obstacles works best at 30 degrees from boresight angles. Note that strong reflectors, such as glass walls, usually can be seen up to almost 90 degrees from boresight. At the core of the detection algorithm is a fast Fourier transform (FFT), which is used to improve the signal-to-noise ratio (SNR) of obstacles in the vicinity of the radar sensor for extraction of angle alpha and distance d.

Four basic steps are repeated in this algorithm:

- Collecting radar sweeps from the IQ Service at a sweep frequency of fs
- Perform FFT of N consecutive sweeps and estimate the power spectral density (PSD)
- Perform peak detection on the PSD
- Report distance d and estimate of angle alpha for found peaks (obstacles)

For further understanding and information we recommend downloading our exploration tool, available from www.developer.acconeer.com.

2 Detecting Obstacles

Detecting obstacles is a question of how well we can distinguish radar signals reflected off them from the noise floor of the sensor. While the sensor has very well-defined radar emission properties, radar scattering and reflection off obstacles depends strongly on their individual physical properties. Hence, the detection probability for obstacles is tied to the following key characteristics:

- distance of obstacle to radar sensor
- angle of obstacle to radar sensor
- reflectivity of obstacle at 60.5 GHz given by shape and material

In particular, certain materials are known to only weakly reflect radar, while others are excellent reflectors. Untreated wood followed by treated wood and some plastic types fall in the first category and thus the maximum distance at which obstacles made from these materials can be detected is reduced. Metal, glass and water on the other hand are very good reflectors and can often be detected at quite far distances and large angles. Most other materials have reflectivity properties in between those two groups (most plastic types, tiles, granite) can be detected reasonably well at normal distances. Since radar emission is far outside the visible region of the EM-spectrum, color of materials does not matter.

It is important to understand, that radar signals may reflect off flat surfaces like a window or door in the same way visible light reflects off a mirror. Thus, if the obstacle has a large flat surface, which has a shallow angle with respect to the radar sensor normal, the obstacle might not be detected very well even if it is made from well reflecting material.

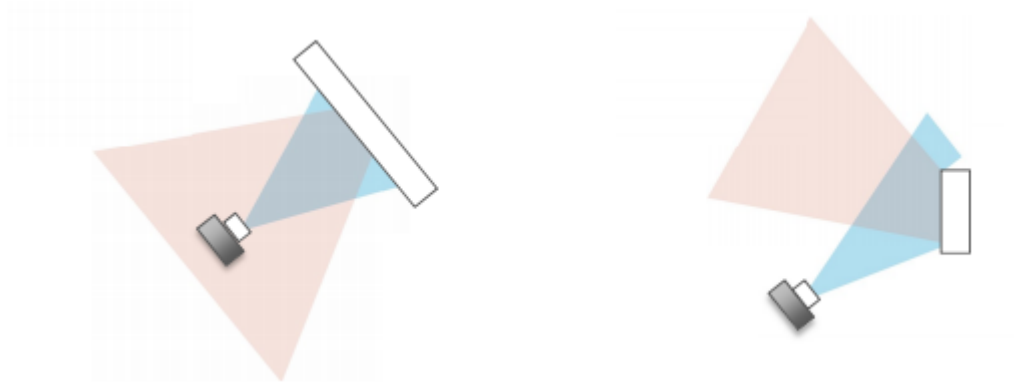
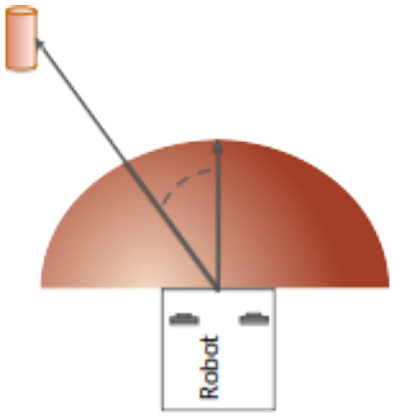


Figure above show object with normal incidence resulting in good reflectivity and object with oblique incidence resulting in reduced detectability

3 Understanding the Results

The obstacle detector reports obstacles with 3 parameters, distance d, angle alpha and amplitude A. The amplitude is a measure of how much signal was reflected for that specific object; a value that is important for setting optimized thresholds for the obstacle detector.



The distance is the actual distance of the obstacle to the radar sensor and the angle is the measured with respect to the direction of motion of the radar sensor. If you need the “normal” distance of the obstacle to the sensor, you can calculate it via the relation cos(alpha)d.

It is important to keep in mind that the reported angle is only reliable if the radar is moving with a known velocity and the object is at rest. For scenarios, where both, the sensor/robot and the object are moving, no definitive conclusion can be made about the objects true angle. The reason for this is that the angle is only inferred, i.e. calculated from the measured radial velocity of the object and not directly measured.

Since the obstacle’s radial velocity is obtained by performing an FFT over a given number of sweeps, it should be noted that the maximum velocity v that the detector can measure is

$$v_{max} = \frac{\lambda f_s}{4},$$

where f is the sweep frequency and lambda is the radar wavelength (˜5.0 mm). For objects that move at higher velocities, the reported object velocity, and therefore also angle, is incorrect. Thus, it is important to initialize the detector with settings that reflect the expected velocities for the desired application. For example, with a sweep frequency of 250 Hz, objects can be reliably detected at velocities below 30 cm/s. If one knows the radar is mounted on the robot so that object moving in one only direction can be observed, the maximum measurable velocity doubles to 60 cm/s. From v2.7.2, the absolute value of the radial velocity is used in the angle calculation to accommodate a negative radial velocity (as seen for a sensor that is facing the reverse direction of the robot motion). Please note that the sign of the radial velocity is unreliable for an obstacle that is straight ahead of the robot when moving at maximum speed due to frequency folding in the computational algoritm.

The highest possible sweep frequency is depending on the MCU used and the range being scanned each sweep.

3.1 Output from the Detector

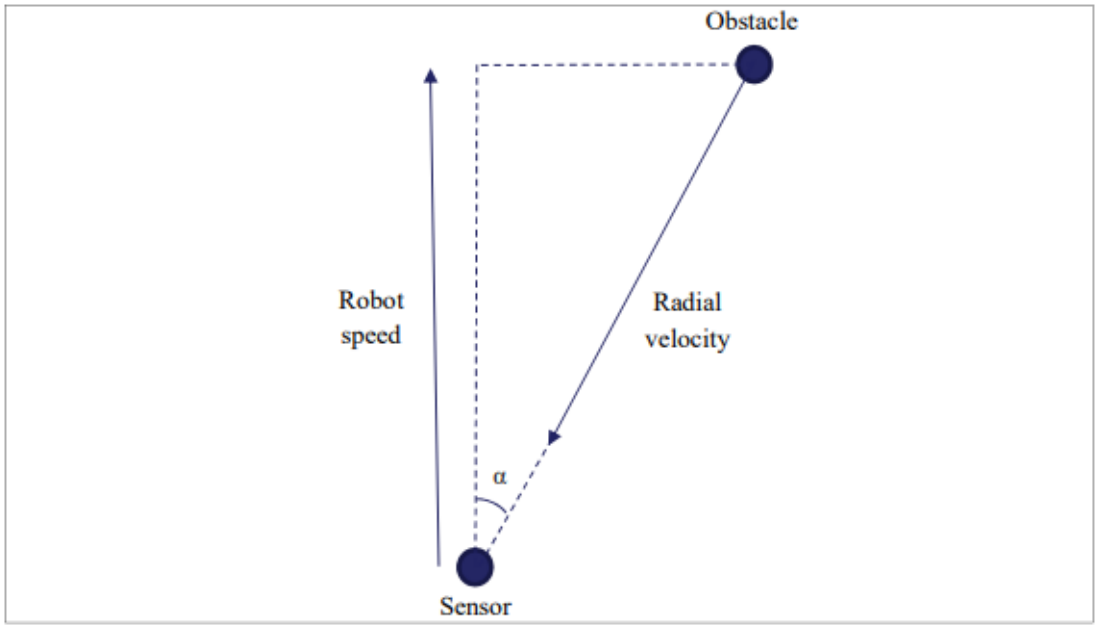
Detections are fetched from the obstacle detector.

```
acc_obstacle_t obstacles[max_number_of_obstacles];
acc_detector_obstacle_t obstacle_data;
bool status = true;

acc_detector_obstacle_result_info_t result_info;

obstacle_data.obstacles = obstacles;

do
{
 status = acc_detector_obstacle_get_next(handle, &obstacle_data, &
    result_info);
}
while (status && !result_info.data_available);
```

Detections will be available in obstacle_data when data_available is set. One obstacle is represented by an acc_detector_obstacle_t struct.

```
typedef struct
{
 float radial_velocity;
 float distance;
 float amplitude;
} acc_obstacle_t;
```

Amplitude is absolute value of complex IQ data and distance is the estimated distance in meters. Radial velocity is the radial velocity towards the sensor.



Radial velocity is used to estimate the angle of an obstacle using

$$\alpha = \cos^{-1}\left|\frac{v_{radial}}{v_{robot}}\right|$$

4 Configuring the Obstacle Detector

4.1 Create an Obstacle Configuration

To start configuring the obstacle detector, an obstacle configuration must be created. The configuration will contain default parameter settings.

```
// Create an obstacle configuation
acc_detector_obstacle_configuration_t configuration =
    acc_detector_obstacle_configuration_create();
```

4.2 Setting General Settings

General settings are similar to the other services and detectors. These settings determine how the ensor will generate data.

4.2.1 Setting Range Start

Configure how far from the sensor the detection range will start.

```
// Set range start to 0.12 meters
acc_detector_obstacle_configuration_set_range_start(configuration, 0.12f);
```

4.2.2 Setting Range Length

Configure how long the detection range will be. Maximum length of detection range will depend on hardware computation power.

```
// Set range length to 0.3 meters
acc_detector_obstacle_configuration_set_range_length(configuration, 0.3f);
```

4.2.3 Setting Sensor

When using a connector board with multiple sensors, configure which sensor should be used by the detector. If multiple sensors should be used, they must be configured separately.

```
// Set detector to use sensor 1
acc_detector_obstacle_configuration_set_sensor(configuration, 1);
```

4.2.4 Setting Gain

Advanced users can control the receiver gain in the sensor. This may be useful for example in cases when measuring reflections from strong reflectors close to the sensor. The gain vale must be between 0.0 and 1.0, where 0.0 is the lowest gain and 1.0 is the highest gain.

```
// Set receiver gain to 0.7
acc_detector_obstacle_configuration_set_gain(configuration, 0.7f);
```

4.3 Setting Detector Specific Settings

4.3.1 Setting Max Number of Obstacles

Configure how many obstacles the detector should look for. The number of obstacles found will affect the calculation time of the algorithm. One physical object may be returned as multiple obstacles, if the detector detects multiple reflections from the same object.

```
// Set max nubmer of obstacles to 1
acc_detector_obstacle_configuration_set_max_number_of_obstacles(
    configuration, 1);
```

4.3.2 Setting Max Speed

This parameter is the speed limit of obstacles moving towards the sensor, i.e. the speed limit of the robot which sensor is mounted on. If moving faster than this, the detector will not be able to properly estimate the angle of the obstacle. The max speed parameter determines in what frequency the sensor will produce data.

```
// Set maxumim speed to 0.3 m/s
acc_detector_obstacle_configuration_set_max_speed(configuration, 0.3f);
```

4.3.3 Setting Allow Reverse

Disabling allow reverse will make it possible to have a higher max speed at a lower frequency. However, the detector will not be able to estimate angle of obstacles moving away from the sensor, i.e. if the sensor is mounted backwards. This setting is recommended if the sensors are mounted in the forward direction.

```
// Enable allow reverse
acc_detector_obstacle_configuration_set_allow_reverse(configuration, true);
// Disable allow reverse
acc_detector_obstacle_configuration_set_allow_reverse(configuration, false);
```

4.3.4 Setting Downsample Scale

Down sampling sensor data will increase the speed of the detection algorithm but may decrease the accuracy of distance and angle estimations.

```
// Set down sample scale to 8
acc_detector_obstacle_configuration_set_downsample_scale(configuration, 8);
```

4.3.5 Setting Edge to Peak Ratio

To improve distance estimation, edge detection can be used. Edge to peak ratio controls the ratio between the peak amplitude and the edge amplitude. The edge amplitude will never be lower than the configured threshold.

```
// Set edge to peak ratio to 0.5
acc_detector_obstacle_configuration_set_edge_to_peak_ratio(configuration,
    0.5f);
```

4.3.6 Setting Distance Offset

Distance offset can be used to remove absolute offset error in distance estimation. Depending on sensor integration, an absolute error can be introduced to the distance estimation. The value will be subtracted from the estimated distance.

```
// Set distance offset to 0.01 meters
acc_detector_obstacle_configuration_set_distance_offset(configuration, 0.01f
    );
```

4.3.7 Setting Range End Over-scan

The sensor will be set to sweep farther than the range. This will improve distance estimation close to the end of the range. If set to 0.03 meters, the sensor will sweep 0.03 meters beyond the end range but obstacle detection will still be done within the range.

```
// Set range end overscan to 0.03 meters
acc_detector_obstacle_configuration_set_range_end_overscan(configuration,
    0.03f);
```

4.3.8 Setting Speed Filter High Pass

Objects that have a radial speed towards the sensor lower than the highpass masking speed are not reported in the output. Set the high-pass masking for radial speed. The value represents the center of the gradient transition from zero amplitude to unchanged amplitude.

```
// Set highpass masking for radial speed to 0.01 m/s
acc_detector_obstacle_configuration_set_speed_filter_highpass(configuration,
     0.01f);
```

4.4 Setting Background Estimation

In background estimation and cancellation, the average signal power for all different speed indexes are recorded during the estimation and later subtracted from the signal.

The number of iterations that should be used for background estimation is configurable. Setting to zero will disable background estimation. A higher number of iterations will improve the background estimation.

```
// Set background estimation iterations to 10
acc_detector_obstacle_configuration_set_background_estimation_iterations(
    configuration, 10);
```

4.5 Setting Thresholds

By configuring amplitude thresholds in the detector, it is possible to control the sensitivity of the detector. The thresholds should be set high enough to avoid getting false detections caused by noise, and low enough to not miss detections from small objects.

Close to the sensor, we have the static reflections from sensor casing and floor. Here we use the stationary threshold. It makes sure that stationary reflections are not reported as object detections.

We use the moving threshold for objects that moves towards or away from the sensor. We also use the moving threshold for objects that are further away from the sensor than distance_limit_far. Above this distance we assume that there is no contribution from stationary clutter caused by stationary objects. This makes it possible to detect objects at angles close to 90 degree if they are above distance_limit_far.

```
// Set amplitude thresholds
const acc_detector_obstacle_configuration_threshold_t thresholds =
{
 .stationary = 0.05f,
 .moving = 0.02f,
 .distance_limit_far = 0.25f
};
acc_detector_obstacle_configuration_set_thresholds(configuration, &
    thresholds);
```

5 Running Obstacle Detector

5.1 Creating and Activating the Detector

When a configuration has been created and configured, the detector can be created. The configuration is validated, and memory needed by the detector is allocated during creation.

```
// Create an obstacle detector
acc_detector_obstacle_handle_t handle = acc_detector_obstacle_create(
    configuration);
if (handle == NULL)
{
 /* Handle error */
}
```

If detector creation returns NULL, an error has occurred. Otherwise the detector is ready to be activated.

```
// Activate an obstacle detector
bool status = acc_detector_obstacle_activate(handle);
if (!status)
{
 /* Handle error and destroy obstacle detector */
}
```

When the detector is activated it will be possible to start retrieving obstacles from the detector.

5.2 Deactivating and Destroying the Detector

To stop measurements, the detector must be deactivated.

```
// Deactivate an obstacle detector
bool status = acc_detector_obstacle_deactivate(handle);
if (!status)
{
 /* Handle error and destroy obstacle detector */
}
```

When the detector is stopped it can be destroyed to deallocate resources used by the detector.

```
// Destroy an obstacle detector
acc_detector_obstacle_destory(&handle);
```

6 Tuning the Detector

In order to achieve the best possible detector performance, it is important to conduct proper tuning of the detector for each application and hardware integration. There are several parameters that require tuning.

Thresholds for obstacle detection: if the amplitude of a found obstacle is below the threshold it will be discarded.

Background estimation: it is usually necessary to perform a background estimation as sensor enclosure, housing and surroundings of the sensor may generate reflections. The better the background estimation, the lower thresholds can be used.

For both threshold tuning and background estimation, Acconeer offers an easy-to-use Python based graphical user interface (available upon request).

6.1 Background Estimation

The detector offers a method for background estimation and cancellation. Measurements need to be performed with empty space in front of the sensor, but with full enclosure and housing.

For most cases, background estimation and cancellation is recommended, as it allows to set lower thresholds. With this method we estimate a full FFT background map. This map is subtracted from each FFT calculated, when the detector is running. For estimating a good background map, we recommend using 10 iterations, which means collecting $16 \cdot 10 = 160$ sweeps.

6.2 Thresholds

Setting too low thresholds will result in an increased number of false detections, while setting too high thresholds will result in reduced detector sensitivity and increased number of missed objects. Thus, finding the right thresholds is very important.

In the most basic setup, two different thresholds need to set, one for moving objects and one for static objects. The threshold for static objects is typically higher since enclosure, housing and surroundings are seen as static objects and need to be removed properly. Thresholds for moving objects, on the other hand, can be much lower.

For a more detailed explanation of the different thresholds that can be set we would like to refer you the available tuning tool and its documentation.

7 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.