Speed Detector

Contents

Introduction

A radar sensor is a natural fit for different types of speed measurements. While the traditional use case is traffic monitoring, we can with the A121 measure the speed of small objects close to the sensor, while still having the possibility to measure the speed of objects several meters away.

The algorithm described in this section measures the speed of an object at a set of specified distances (typically a single distance). It is capable of determining the direction of movement (away from or towards the sensor) as well as having a dynamic precision, which is at a trade-off with the update rate (higher precision yields a lower update rate).

How strong reflection the object needs to have in order to be measured is set by a threshold parameter. This is highly use case specific and needs to be manually adjusted.

latest

Theoretical concept

The concept behind this algorithm is similar to how traditional radar measures speed using the pulse doppler effect - a high amplitude frequency component correspond to a specific speed of the measured object. If we consider the case where we sample at a single distance point, we note that small movements within the time of a frame will correspond to phase differences in the received signal. See fig. Figure 19. Since the sweep frequency determines the time (T_s) between the sampled points, we can obtain a speed by looking at the frequency domain of the sampled signal. The signal is converted to the frequency domain using a segmented version of the Discrete Fourier Transform (DFT), called Welch's method.

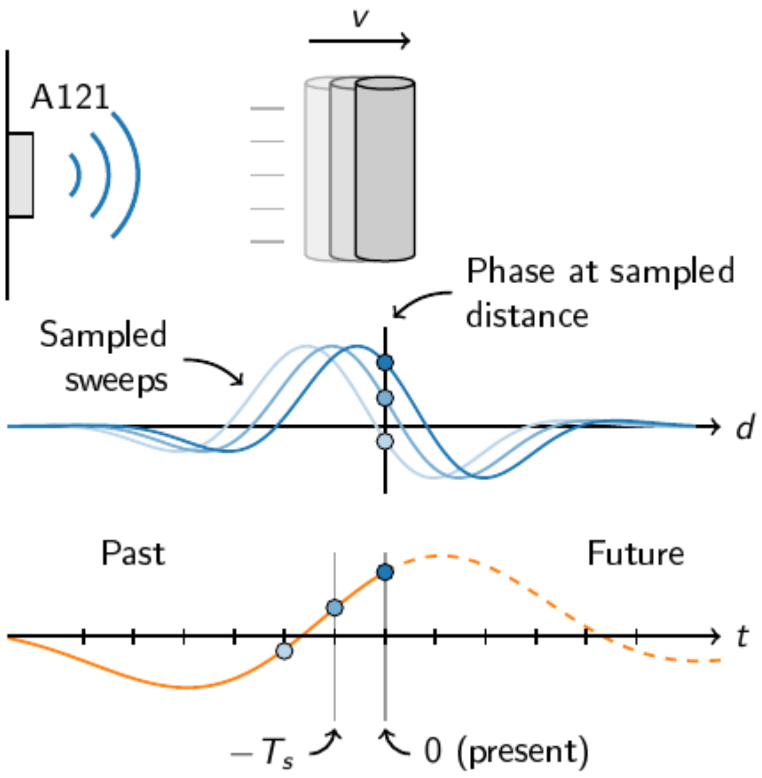


Figure 19 How phase changing over time can be interpreted as a frequency.

The DFT is calculated on the sweeps in each frame, so the `num_bins` directly correlates to how many frequency bins we have in the DFT. Therefore, the `num_bins` variable naturally lets us set the resolution of the output values. A good model to imagine is that we divide the [-`max_speed`, `max_speed`] interval into a number of evenly spaced points, corresponding to the `num_bins` parameter, these points defines which values can be reported by the detector.

As an example, assume that we have a `max_speed` of 10 m/s and `num_bins` is set to 3 (which is a very low value). This means that the only speeds that can be reporte latest
-10 m/s, 0 m/s and 10 m/s, where the sign indicates whether the object is moving towards or away from the sensor.

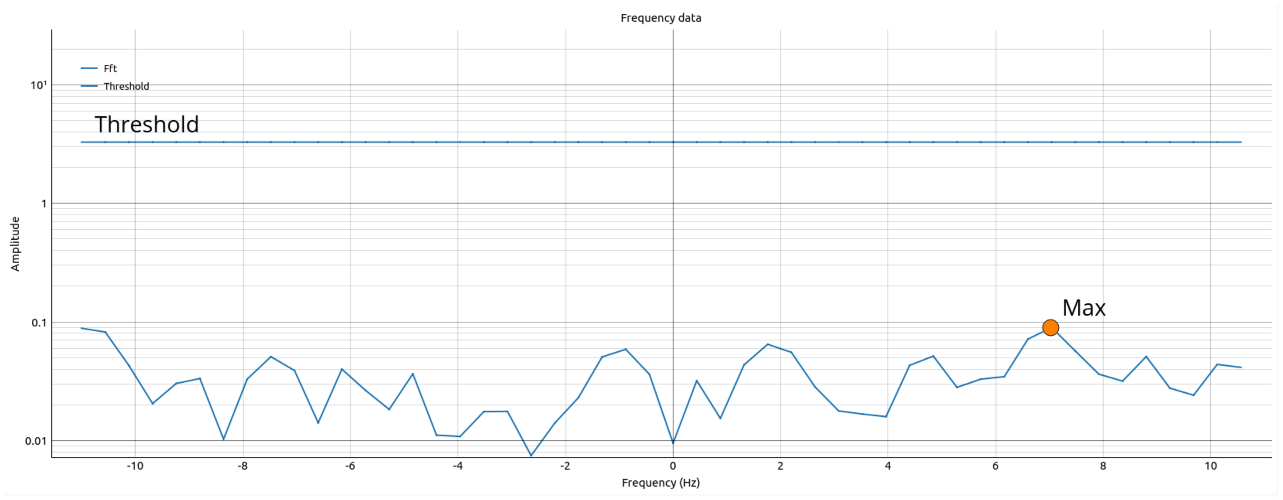
Implementation

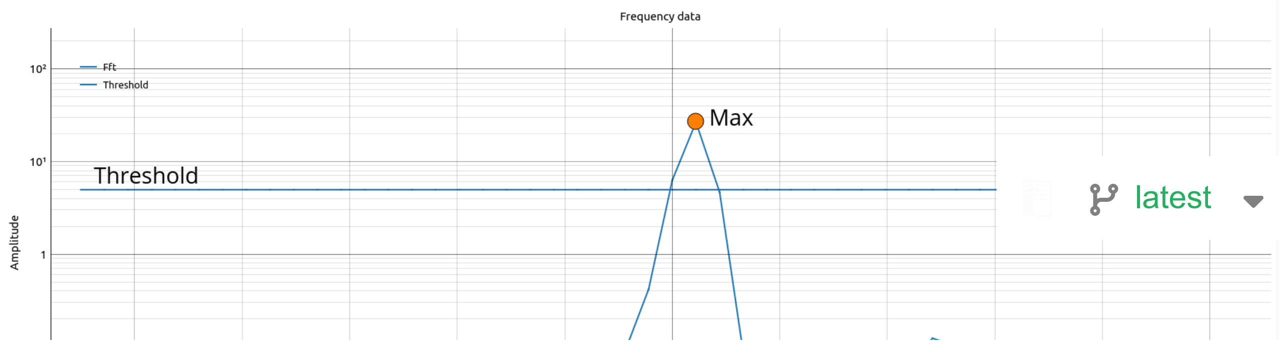
Welch’s method

The DFT is implemented using Welch’s method directly on each frame with no overlap (then also called Bartlett’s method) and a segment length of 1/4 of the number of sweeps. This is already accounted for in the adjustment of the parameter “`num_bins`”, so the number of samples will actually be 4 times `num_bins`. For example, if we set `num_bins` to 300, each frame will consist of 1200 sweeps, with a sweep rate depending on the `max_speed` parameter.

Threshold

The frequency component with the highest amplitude is compared against a threshold based on the median value of the whole frequency spectra. If the max value is above the threshold, the point is interpolated using the neighboring points to find a max value. The threshold is computed by taking a constant multiplied with the median value. In Figure 20 we use a threshold value of 100, taken from the default preset in the exploration tool.





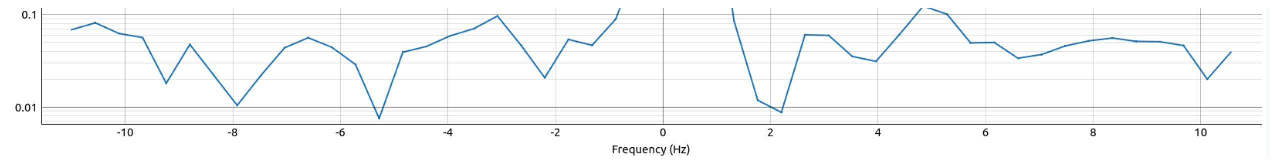


Figure 20 Two examples of where the max value is below the threshold and above the threshold. In this example, the threshold is set to be 100 times the median value. Note the log scale on the y-axis.

Interpolation

The reported value is also interpolated with both neighboring values to get an estimate of the actual peak location. This is the value that is reported back as the speed for the depth.

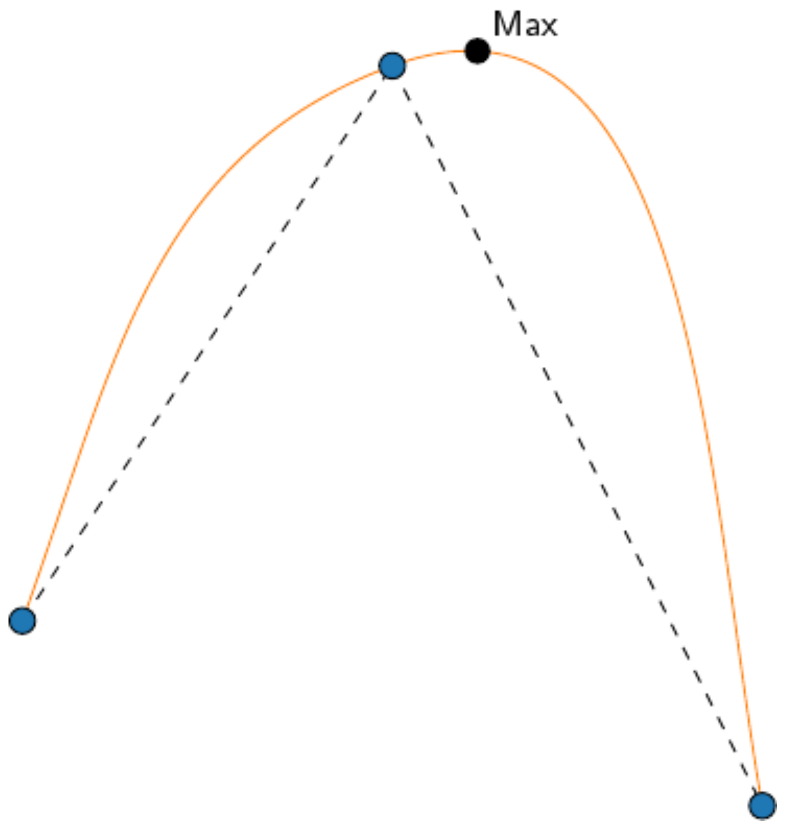


Figure 21 How the maximum can be located outside the found points. The orange line illustrates the interpolated value.

latest

Parameter Guide

Several of the detector parameters are automatically configured, unless the override box is ticked. It is recommended to use the automatic configuration as a starting point. This section describes some more detailed aspects of configuration that might be of interest. Under the "Current sensor settings" foldout, the settings that will be used in the sensor given the current detector configuration can be inspected. Most erroneous configurations will be caught by errors and warnings in the interface, but it is possible to configure the detector in such a way that the warning will be shown as an exception when trying to start.

Number of points

> If the `num_points` parameter is increased to a higher value than 1, a warning will appear that indicates that more than one depth is measured and that we cannot be sure which object is reported. The speed estimation will be done separately on each depth, and then the maximum value will be reported in ET. However, all values can be found in the DetectorResult that is returned, it is only for the display history that the maximum value is selected.

Step length

> The step length is automatically configured to be roughly the same as the pulse length, which is dependent on the profile. This setting can be increased without any issues. If we lower the step length to less than the default value, we have an overlap of what is "seen" by the sensor.

Sweep rate

> The sweep rate is a function of the max measurable speed (and vice versa), this should not be adjusted, instead, adjust the max speed.

Frame rate

latest

> This is not automatically set anywhere, since it is dependent on the sweeps per frame,

> sweep rate and HWAAS, it will be adjusted to the max possible value according to that. As with the sweep rate, it is recommended to not adjust this manually.

HWAAS

> The HWAAS parameter is automatically adjusted to be as high as possible, while still within the timing constraints imposed by the number of sweeps and profile. This can be lowered from the automatic value, which will reduce power consumption, but adjusting this will not change the sweep rate nor the frame rate.

Number of bins

> This parameter is very important for the performance of the detector. A high value will allow for better precision of the output, the trade-off is that more bins will require more sweeps to be collected, which will affect the HWAAS and frame rate. There is also a hard limit on the number of bins that is set by a buffer size on the chip.

Max speed

> The other main parameter to adjust, this defines the max speed that is detectable by the detector. The interpolation makes this a truth with modification, but in general, this should be seen as a limit on what speed the detector can detect. Set this to a speed that is slightly higher than what you expect to measure, just setting it as high as possible will likely yield an incorrect and sluggish behavior.

Detection threshold

> The detection threshold is what prohibits the detector from reporting random deviations as detected speeds. This is described in the section of implementation details. An interesting note is that since the amplitude of the frequency is correlated to the amplitude of the reflected signal, we can filter out lesser reflective moving objects by increasing the threshold. This parameter should be adjusted by ex latest
> the intended use case.

Configuration Guide

There are two presets: *default* and *traffic*.

Default

Default is intended to be a good setup for having the detector at your desk and confirming functionality by waving in front of it. Notice that the high threshold value can be increased another factor 10 without much difference to the performance. If we increase the threshold even further, until we no longer detect any speeds, we can instead use a stronger reflector (perhaps a smartphone) and regain detection. We then start measuring speeds on the movement of the reflector, but not anything else. This idea of adjusting the threshold to only detect the objects of interest allows us to use the speed detector as a counter of moving objects.

Traffic

Continuing with the other preset, *traffic*. Here we have a longer range, which almost certainly will need adjustment when looking at a road. When aiming the sensor at cars passing by, we notice that we can not only measure the speed, but also that we see a gap between measurements. Combining the information of the time gap between *new* detections, with the speed of the detected object we can also get a good estimate of the distance between the cars.

Another thing to notice when aiming the sensor at a road is that we only measure the speed component towards the sensor. If we have an angle between the sensor and the road (which is usually the case, since measuring straight ahead would require a position on the road), we will observe a lower speed that the object is traveling at. See more information here: Cosine error effect.

Further reading: welch method,

Configuration parameters

latest

class **acconeer.exptool.a121.algo.speed._detector.DetectorConfig**(**, start_point: int = 200, num_points: int = 1, step_length: int | None = None, profile: Profile | None = None, frame_rate: float | None = None, num_bins: int = 50, sweep_rate: int | None = None, hwaas: int | None = None, max_speed: float = 10.0, threshold: float = 100.0*)

start_point: *int*

Start point of measurement interval.

num_points: *int*

Number of points to measure.

step_length: *int | None*

Step length between points.

profile: *Profile | None*

Sets the profile. If no argument is provided, the highest possible profile without interference of direct leakage is used to maximize SNR.

frame_rate: *float | None*

Frame rate in Hz.

num_bins: *int*

Determines the resolution in m/s by max_speed/num_bins.

sweep_rate: *int | None*

Sweep rate in Hz.

latest

hwaas: *int | None*

Number of HWAAS.

max_speed: *float*

Max detectable speed in m/s.

threshold: *float*

Peak relative height to median scaled PSD. E.g., 10.0 indicates that we need to have 10 times the median value to trigger.

Detector result

class `acconeer.exptool.a121.algo.speed._detector.`**DetectorResult**(*, *speed_per_depth: ndarray[tuple[int, ...], dtype[float64]]*, *extra_result: DetectorExtraResult*)

speed_per_depth: *ndarray[tuple[int, ...], dtype[float64]]*

The measured speed for each depth

extra_result: *DetectorExtraResult*

Detailed results, used for plotting

latest