

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE, FAKULTA ELEKTROTECHNICKÁ

**Diplomová práce**

**Velocity Measurement by Doppler Radar**

**Dopplerovský radar pro měření rychlosti – funkční vzor**

Studijní program: Elektronika a komunikace  
Studijní obor: Rádiové systémy  
Vedoucí práce: prof. Ing. František Vejražka, CSc.



## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Oravec** Jméno: **Matej** Osobní číslo: **437396**  
Fakulta/ústav: **Fakulta elektrotechnická**  
Zadávající katedra/ústav:  
Studijní program: **Elektronika a komunikace**  
Specializace: **Rádiové systémy**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Dopplerovský radar pro měření rychlosti – funkční vzor**

Název diplomové práce anglicky:

**Velocity Measurement by Doppler Radar**

Pokyny pro vypracování:

V diplomové práci vytvořte experimentální zařízení mající povahu funkčního vzorku a upravte ho tak, aby bylo využitelné pro pokusy v terénu bez dalších pomůcek. V přiměřeném rozsahu se věnujte software zařízení a přehledně pojednejte o interpulsové modulaci a modulaci sledu impulsů a o vlivu modulace na parametry radaru. Rozsah práce: max. 100 stran A4 a příp. jednoduchá výkresová dokumentace. Výsledky pokusů, oscilogramy, spektrogramy apod.

Seznam doporučené literatury:

[1] RICHARDS, Mark A. Fundamentals of Radar Signal Processing. 2nd ed. New York: McGraw – Hill, c2014. ISBN 978-0-07-179832-7. JANKIRAMAN, Mohinder. FMCW radar design. Boston: Artech House, [2018], ©2018.. xviii, 401 pages. Artech House radar series. ISBN 978-1-63081-567-7.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**prof. Ing. František Vejražka, CSc., katedra radioelektroniky FEL**

Jméno a pracoviště druhého(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: \_\_\_\_\_ Termín odevzdání diplomové práce: \_\_\_\_\_

Platnost zadání diplomové práce:

**do konce letního semestru 2020/2021**

prof. Ing. František Vejražka, CSc.  
podpis vedoucí(ho) práce

\_\_\_\_\_ podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

\_\_\_\_\_ Datum převzetí zadání

\_\_\_\_\_ Podpis studenta





## **Abstract**

Employing radar as an onboard speedometer could be advantageous when compared to more widespread methods based on a wheel diameter or GNSS. It offers functionality in environments where satellite signal is not available, and its performance is not influenced by wheel slipping, or development of a wheel diameter in time. In the scope of this thesis, a functional FMCW radar speedometer based on a commercial FMCW evaluation module is designed, realized, and tested. The theory behind FMCW radar is examined and based on that, the most suitable signal processing algorithm is developed. The resulting device is tested in various environments, and its performance is reviewed.

## **Abstrakt**

Použitie radaru ako palubného merača rýchlosti má oproti bežne využívaným metódam, založeným na priemere kolesa či satelitných systémoch určovania polohy, množstvo výhod. Umožňuje fungovanie i vtedy, keď satelitný signál nie je dostupný, a jeho fungovanie nie je ovplyvnené preklzovaním kolies či zmenami ich priemeru v čase. V rámci tejto práce je navrhnutý, realizovaný a testovaný radarový merač rýchlosti, založený na komerčnej FMCW radarovej doske. Je zhrnutá teória FMCW radaru a na jej základe je navrhnuté spracovanie signálu a použité zapojenie. Výsledné zariadenie je testované v rôznych podmienkach a nezanedbateľná časť je venovaná jeho presnosti.

## **Prehlásenie**

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavání etických princípov pri príprave vysokoškolských záverečných prác.

V Prahe 13. mája 2020

---

Podpis študenta

## **Pod'akovanie**

Moje veľké ďakujem patrí pánovi profesorovi Vejražkovi za mnohoročnú podporu, cenné rady a úžasný ľudský prístup. Vaškovi Navrátilovi za jeho nesmierne vedomosti, ktoré mi často poskytoval formou skutočne dobre mierených rád. A v neposlednom rade mojej rodine, ktorá je pre mňa obrovskou oporou v každej situácii.

---

# Contents

<b>Contents</b>	<b>iv</b>
<b>1 Doppler radar</b>	<b>1</b>
1.1 History	1
1.2 Basic principles	1
1.2.1 Doppler shift	1
1.2.2 Radar species	2
1.3 FMCW radar	3
1.3.1 Chirp signal	3
1.3.2 Range estimation	4
1.3.3 Velocity estimation	7
1.3.4 Range – Doppler processing	8
1.3.5 Multiple inputs, multiple outputs (MIMO)	9
1.3.6 Angle estimation	11
1.4 Radar sensor geometry	12
<b>2 Hardware and its limitations</b>	<b>14</b>
2.1 Market options	14
2.2 The radar board	14
2.2.1 Components of the board	15
2.2.2 Antenna pattern	16
2.2.3 Far field	17
2.2.4 Area of reflection	18
2.3 Computer subsystem	24
2.3.1 Raw ADC samples	24
2.3.2 Decentralized processing	26
2.4 Power supply	27
<b>3 Signal and its processing</b>	<b>29</b>
3.1 The original firmware	29
3.1.1 Flashing the firmware to the board	29
3.1.2 Configuration via the CLI	30
3.1.3 Received data format	40
3.2 Algorithms for velocity evaluation	41
3.2.1 Mean of Doppler bins argmax	41
3.2.2 Mean of spectral centroid of Doppler vectors	42
3.2.3 Mean Doppler of detected objects	43
3.3 Chirp train design	44

3.3.1	Start frequency of the chirp . . . . .	44
3.3.2	Idle time . . . . .	45
3.3.3	ADC start time . . . . .	47
3.3.4	Ramp time . . . . .	47
3.3.5	Frequency slope . . . . .	47
3.3.6	Peak to average power ratio in range – Doppler matrix . . . . .	48
3.3.7	Model 1 . . . . .	48
3.3.8	Model 2 . . . . .	52
3.3.9	Selecting the best configuration . . . . .	54
3.4	Velocity evaluation . . . . .	57
3.4.1	Measurement methodology and setup . . . . .	57
3.4.2	Analysis of measured data . . . . .	59
3.4.3	Procedure of velocity evaluation . . . . .	61
3.4.4	Quantifying accuracy and precision . . . . .	61
3.4.5	Stability of measurements, filtering results . . . . .	69
3.4.6	Measuring velocity of a car . . . . .	69
3.4.7	Measurement of bias . . . . .	70
<b>4</b>	<b>Software implementation</b>	<b>72</b>
4.1	Backend modules . . . . .	72
4.1.1	Data parser . . . . .	72
4.1.2	Range – Doppler matrix processor . . . . .	74
4.1.3	Bluetooth communication module . . . . .	75
4.1.4	Utility module . . . . .	75
4.2	Front end . . . . .	76
4.2.1	The Plotter module . . . . .	76
4.2.2	Velocity viewer GUI . . . . .	80
4.3	Documentation . . . . .	82
<b>5</b>	<b>Conclusion</b>	<b>83</b>
<b>A</b>	<b>UART data structure</b>	<b>85</b>
<b>B</b>	<b>Range – Doppler heatmap plots</b>	<b>88</b>
B.1	Adjustment of idle time . . . . .	88
B.2	Adjustment of ramp time . . . . .	89
<b>C</b>	<b>Selected source codes</b>	<b>90</b>
C.1	PAPR modeling . . . . .	90
C.2	Other . . . . .	91
	<b>Acronyms</b>	<b>92</b>
	<b>Bibliography</b>	<b>94</b>



# Doppler radar

Although the word *radar* stands for *radio detection and ranging*, present systems offer much more. Not only presence or absence of a target and its distance from the radar could be addressed. Radar systems in general are either capable to resolve velocity of the target, its azimuth and elevation, and—considering outgrowths to the fields of its application—also cloud positions and their respective repletions (meteorology), terrain maps (surveillance), or breath monitoring (healthcare). In this chapter, we briefly overview history of radar, introduce its most used types, and in more detail summarize the working principle of FMCW (frequency modulated continuous wave) radars.

## 1.1 History

The history of radar got birth shortly after the first inventions in the field of electromagnetic theory. In the early 20th century, 20 years after Hertz's demonstration of electromagnetic waves reflection, German inventor Christian Hülsmeyer and American U.S. Naval Research Laboratory (NRL) researchers Taylor and Young experimented with ship detection using EM waves [1]. A patent for the CW (continuous wave) radar has been accepted in 1934. Pulsed radar has been demonstrated for the first time in 1936. As time went by, a range of frequencies used by radar systems became wider, especially to the higher part of the spectrum. In contrast with the first radars working in HF (high frequency) band, there are radar frequency bands up to 110 GHz considered as standard for some radar applications nowadays.

## 1.2 Basic principles

The theoretical basics behind radar differ significantly according to the particular *type* of radar. Nevertheless, the physical principles addressing the propagation of EM waves or the well-known *Doppler principle* are related to all of them.

### 1.2.1 Doppler shift

In general, *Doppler principle* proposes that *frequency of the wave reflected from a moving object is shifted from the original frequency of the incoming wave and the frequency shift is proportional to radial<sup>1</sup> velocity of the moving object*. The principle is relevant for EM, as well as for sonic or other waves. Nevertheless, in this thesis we assume all the waves mentioned to be *electromagnetic*. If we have a transmitted wave of the form

$$s_t(t) = A_t \cos(2\pi f_0 t) \quad (1.1)$$

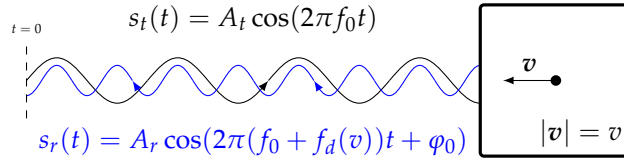
---

<sup>1</sup>Related to the transmitter.

and it reflects back from an object with velocity  $v$  in a direction parallel with the direction of  $s_t$  propagation, its frequency will be changed and the reflected wave could be described as

$$s_r(t) = A_r \cos(2\pi(f_0 + f_d(v))t - \varphi_0). \quad (1.2)$$

In (1.1) and (1.2),  $A_t$  and  $A_r$  are respective amplitudes of the *transmitted* (incoming) and *reflected* waves,  $f_d$  is so-called *Doppler frequency*,  $f_0$  is frequency of the transmitted wave, and  $\varphi_0$  is an initial phase of the reflected wave. The situation is depicted in Fig. 1.1.



**Fig. 1.1:** Doppler effect. Frequency of the reflected wave differs from frequency of the incoming wave. Different amplitudes of the waves are caused by losses in the reflector.

In (1.2), the Doppler shift  $f_d$  is just a *frequency difference*, not absolute frequency. Generally, for the situation from Fig. 1.1, it is

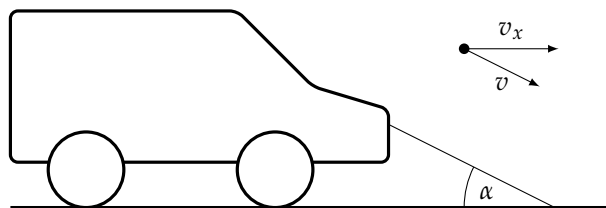
$$f_d = \left(\frac{c_0 + v}{c_0} - 1\right)f_0 \quad (1.3)$$

with  $c_0$  being velocity of light in vacuum (air).

Many other constellations than the one in Fig. 1.1 are possible. For example, the situation when a radar (assume the *monostatic*<sup>ii</sup> one) is placed directly on a moving object and the object travels on the flat ground. Simplistically, that models a situation of a road vehicle which horizontal velocity we would like to know. It is drawn in Fig. 1.2. Assuming an infinitely narrow radar beam and perfectly flat ground, for the Doppler shift in this case it holds [2]:

$$f_d = \frac{2v_x f_0}{c_0} \cos(\alpha), \quad (1.4)$$

where  $v_x$  is the horizontal velocity ( $v$  is its component parallel with radar waves) and  $\alpha$  is the depression angle.



**Fig. 1.2:** Geometry of radar measurement of the vehicle's velocity.

## 1.2.2 Radar species

There are multiple types of radar devices. Particular devices may be categorized according to frequency, number of antennas, their positioning, source of the radar response, etc. The very basic division of radar systems is as follows.

<sup>ii</sup>Monostatic radars have their transmitting and receiving antennas placed very near to each other (it could be even only one antenna serving as TX or RX antenna using time division multiplexing, or a circulator).



**Continuous wave vs. pulsed.** CW systems' measurements are based on observing changes in parameters of continuously-transmitted waveform. On the other hand, the *pulsed* systems use distinct pulses, between which no signal is transmitted.

**Monostatic vs. bistatic.** Monostatic systems have their transmitting (TX) and receiving (RX) antennas collocated, whilst in bistatic systems the antennas are distinct, with their mutual distance comparable to the distance between one of them and the target. There are also *multistatic* systems present, which use multiple TX and RX antennas. Such systems automatically fall within the scope of so-called MIMO (multiple input, multiple output) systems.

**Active vs. passive.** The word *radar* implicitly names an *active* system. That means, it transmits its own waveform. On the other hand, there are also passive systems which only receive the signals of extraneous services. Such systems are sometimes called *passive radars*<sup>iii</sup>.

FMCW radar as the core of this thesis could cause a bit of confusion when talking about its categorization. The main problem with it is that even if it transmits a signal continuously, the signal itself consists of many repeating pulses called *chirps*. Nevertheless, in general, FMCW radars are accepted as continuous wave radars. The rest of its categorization is much more clear – it is *monostatic*, MIMO, and *active* system.

## 1.3 FMCW radar

One special category of radar systems is FMCW (Frequency-modulated, continuous-wave). These systems are of sudden interest, which is mainly credited to the boom of autonomous vehicles in last decade. Even though the major advantages of FMCW systems cover resistance to interception or good range resolution [3], the reason why we have chosen it as the suitable system for our application is simply the *availability* of 77 GHz radar sensors likely to constitute the base of our device in this category on the market.

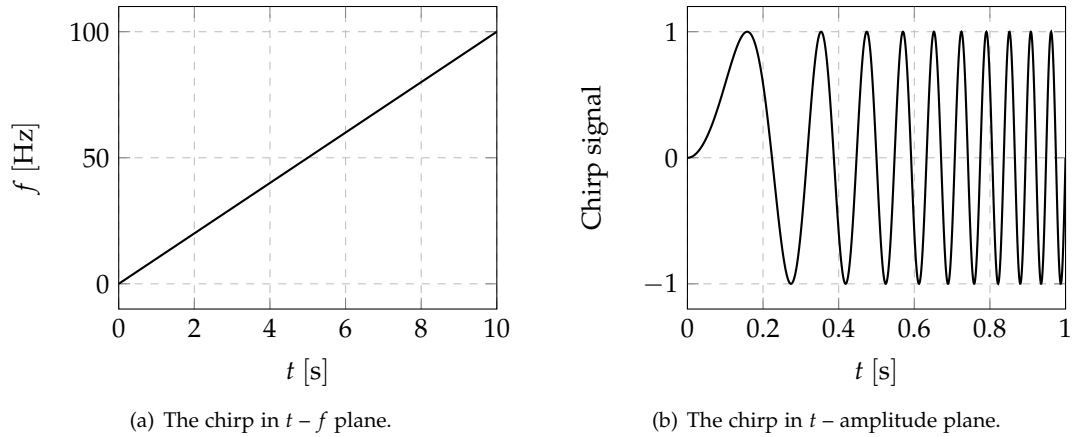
### 1.3.1 Chirp signal

FMCW radars are not only able to provide for *velocity* measurement. Thanks to the frequency modulation used, they could provide for rather precise *range* estimation of the objects in their field of view. Moreover, if the system had more than one RX antenna, it could also give us an information about the *angle* of the targets, which makes the FMCW radars a category of quite universal systems.

The frequency modulation of the waveforms could be of many types. The *linear* one is, however, by far the most widely used [3]. Its signal is called *chirp* and its time – frequency and time – amplitude plots could be seen in Fig. 1.3. Properties of the chirp are usually defined by its amplitude  $A$  and *frequency slope*  $S = \Delta f / \Delta t$ , where  $\Delta f$  and  $\Delta t$  are the frequency sweep and the time sweep, respectively.

---

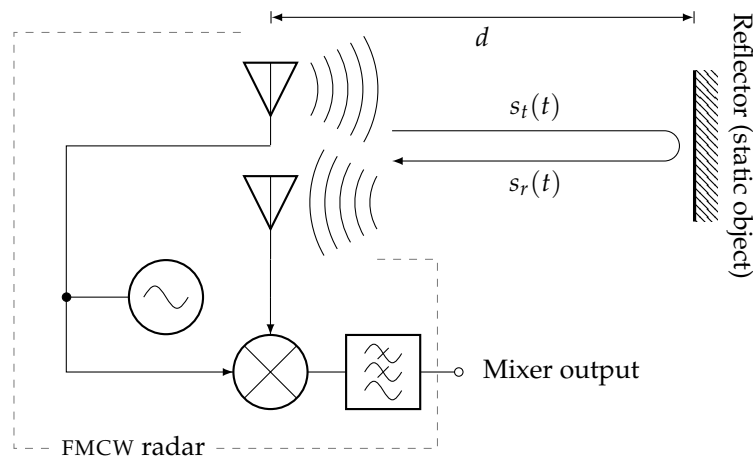
<sup>iii</sup>The passive radars are also referred to as *passive coherent locators*.



**Fig. 1.3:** The course of a linear chirp with  $A = 1$  and  $S = 10 \text{ Hz s}^{-1}$ .

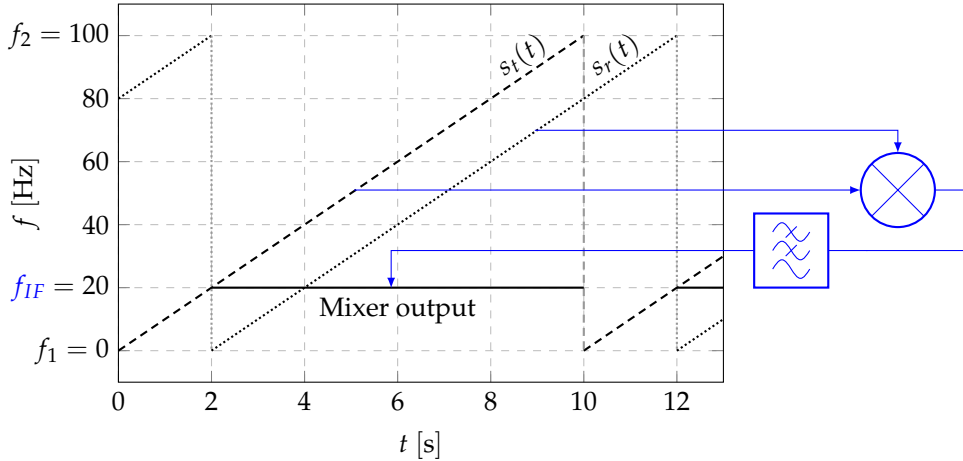
### 1.3.2 Range estimation

Consider the arrangement from Fig. 1.4. After the transmission of a chirp  $s_t(t)$  by the TX antenna, the wave is reflected back from the *static* reflector and the reflected wave propagates towards the radar, in which it is finally absorbed by the RX antenna. Let the reflected signal be denoted as  $s_r(t)$ . Since the material of the reflector is assumed to be non-magnetic, and with higher permittivity in comparison with air (asphalt and concrete both have dielectric constant approximately  $\epsilon_r \approx 10$  [4]), the reflected wave is phase-changed by  $\pi$  rad [5,6]. Furthermore, the reflected wave has smaller amplitude than the incoming one; That is caused by a imperfect reflectivity of the reflector which absorbs a part of energy.



**Fig. 1.4:** An illustrative arrangement of the radar and a reflector.

Let us now focus on *frequency* of the reflected signal. Since the reflector is assumed to be static, no Doppler shift is present and frequency of  $s_r(t)$  is equal to frequency of  $s_t(t)$ . With an assumption of finite chirp (the linear sweeping starts at  $f_1$ , continues until  $f_2$ ;  $f_2 > f_1$  is reached and then restarts immediately at  $f_1$ ), we can draw the following:



**Fig. 1.5:** Time – frequency diagram of the three signals providing for estimation of a distance of a static object.

During the overlap period of ramps of  $s_t(t)$  and  $s_r(t)$ , there is an intermediate frequency (IF) signal present on the output of the mixer – filter cascade. By measuring its frequency  $f_{IF}$ , we are able to appoint the distance from the radar at which the static reflector is as

$$d = \frac{f_{IF} c_0}{2S}. \quad (1.5)$$

For a derivation of (1.5), refer to equations (1.6) to (1.8). Outside the overlap period of the transmitted and the corresponding received ramps, the mixed signal has an unwanted frequency and we have to ensure that the IF signal is only sampled during the overlap period.

It is also useful to know how is the solution affected by a motion of the reflector, i.e. how big is an error in distance estimate  $\Delta d$  for different values of doppler frequency  $f_d$ . Assume the chirp signal to be transmitted to have a form

$$s_t(t) = A_t \cos(2\pi(f_1 + St)t). \quad (1.6)$$

This waveform reflects back from the moving object at the distance  $d$  (similarly as in Fig. 1.1) and in receiver, we obtain signal

$$s_r(t) = A_r \cos(2\pi(f_1 + S(t + \Delta t) + f_d)t + \varphi_0). \quad (1.7)$$

The constant phase shift  $\varphi_0$  could be neglected since it has no influence on frequency of the reflected wave. The stable component<sup>iv</sup> of the mixer output has frequency

$$f_{IF} = f_1 + S(t + \Delta t) + f_d - (f_1 + St) = S\Delta t + f_d = S\frac{2d}{c_0} + f_d, \quad (1.8)$$

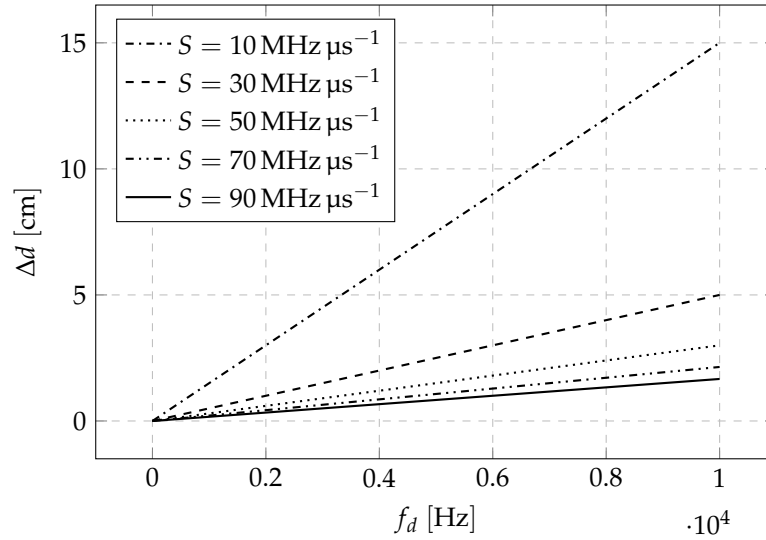
hence the estimated distance of the reflector is

$$d = \frac{(f_{IF} - f_d)c_0}{2S} = \frac{f_{IF}c_0}{2S} - \underbrace{\frac{f_dc_0}{2S}}_{\Delta d} = \text{correct term} - \text{error term}. \quad (1.9)$$

The error in distance estimate ( $\Delta d$ ) depends on the frequency slope of the chirp and—naturally—on the Doppler frequency. Its values for different frequency slope values is depicted in Fig. 1.6. For

<sup>iv</sup>At the output of the mixer, we can divide the signal to the difference part that has a constant frequency, and the sum part having its frequency varying and always higher or equal to the frequency of the difference part. The sum part is to be filtered by the low pass filter. The difference part is the one of our interest and we name it a *stable component*.

sufficiently big  $S$  values, common doppler shifts ( $\approx 10^4$  Hz) do not cause errors in range estimate in less than 5 cm.



**Fig. 1.6:** Error in range estimation  $\Delta d$  caused by nonzero Doppler frequency  $f_d$ .

In equation (1.8), we have used the relationship

$$\Delta t = \frac{2d}{c_0}. \quad (1.10)$$

It is rather intuitive that the time spent by a wave to travel the distance between transmitter and reflector at velocity  $c_0$ , reflect, and then return back at the same speed<sup>v</sup> is exactly  $2d/c_0$ . If the reflector is static, during the time interval of  $\Delta t$ , frequency of the chirp changes by

$$f_{IF} = \Delta f = \frac{\Delta f}{\Delta t} \Delta t = S \Delta t = \frac{2Sd}{c_0}. \quad (1.11)$$

From that, with known  $S$ , the range value could be calculated as

$$d = \frac{f_{IF} c_0}{2S}. \quad (1.12)$$

If there are two or more static reflectors at the same distance from the radar, we clearly cannot distinguish between them using the practice mentioned before. The peaks in the frequency spectrum of the IF signal related to them will cover each other. Assume the overlap period of RX and TX chirps to be  $T$ , and sampling of the IF signal with a sampling period  $T_p$ . The discrete spectrum of the IF signal will be  $(1/T_p)$ -periodic and one period of it will have the same number of samples as the input had:  $T/T_p$  [7]. The frequency difference between two adjacent spectral samples hence will be

$$\Delta f = \frac{1/T_p}{T/T_p} = \frac{1}{T} \quad (1.13)$$

which is the resolution of IF signal's spectrum. This resolution could be converted to the range resolution  $\rho_d$  using (1.12) and (1.13) as

$$\rho_d = \frac{c_0 \Delta f}{2S} = \frac{c_0}{2ST} = \frac{c_0}{2B'} \quad (1.14)$$

<sup>v</sup> Absolute value of velocity.

where  $B$  is a bandwidth of the chirp within the overlap period. In Fig. 1.5, its value is  $B = 80$  Hz. The value of  $\rho_d$  tells us how big should be the range difference between two static objects to be resolved by the radar.

Since *discrete* processing is assumed to be used, in order to fulfil the sampling theorem  $1/T_p \geq f_M^{\text{vi}}$ , we have to ensure

$$d \leq \frac{c_0 f_M}{2S} = \frac{c_0}{2ST_p}. \quad (1.15)$$

The expression on the right side of (1.15) stands for *maximum discernible range* for a particular chirp setup.

### 1.3.3 Velocity estimation

While with cw radar velocity estimate is based directly on observing the spectral content of IF signal, in FMCW radars the principle is slightly different. Let an object with non-zero velocity towards the radar be present in its field of view. Its average velocity in a time interval  $\Delta t$  is

$$v = \frac{\Delta d}{\Delta t} \quad (1.16)$$

where  $\Delta d$  is a spatial shift of the object during  $\Delta t$  measured on the line connecting the radar and the object. Assume that  $\Delta d$  is smaller than the range resolution proper to the chirp used, so if another—let us say static—target was present at the same distance, we would not be able to distinguish the two. However, if the *train* of chirps is transmitted (Assume infinitely-fast transitions like in Fig. 1.5), we could use *phase* of two adjacent chirps to resolve them in a velocity domain.

During the time interval  $\Delta t$ , the motion of object with velocity  $v = \Delta d/\Delta t$  causes a phase shift of the IF signal

$$\Delta\phi = 2\pi \int_0^{\Delta t} f \, dt = 2\pi \int_0^{\Delta t} (f_1 + St) \, dt = 2\pi(f_1\Delta t + S\frac{\Delta t^2}{2}). \quad (1.17)$$

According to (1.10), we can write  $\Delta t = 2\Delta d/c_0$  and further expand (1.17) as

$$\Delta\phi = 2\pi \left( f_1 \frac{2\Delta d}{c_0} + \frac{S}{2} \frac{4\Delta d^2}{c_0^2} \right) = \frac{4\pi f_1 \Delta d}{c_0} + \frac{2S\Delta d^2}{c_0^2}. \quad (1.18)$$

Since the time interval between two adjacent chirps is exactly  $T_c$ , the moving object travels distance  $\Delta d = vT_c$  between them. From that, we obtain a quadratic equation for velocity  $v$ :

$$\frac{2ST_c^2}{c_0^2} v^2 + \frac{4\pi f_1 T_c}{c_0} v - \Delta\phi = 0, \quad (1.19)$$

with roots

$$v_{1,2} = \frac{c_0 f_1 \pi}{f_1 - f_2} \pm \frac{\sqrt{2} c_0 \sqrt{T_c (2T_c f_1^2 \pi^2 + \Delta\phi (f_2 - f_1))}}{2T_c (f_2 - f_1)} \quad (1.20)$$

One of the roots is always fairly negative and we can neglect it. After substitution of common values<sup>vii</sup> for the chirp variables given by Tab. 1.1, we could see that the results for velocity  $v$  using (1.20) are almost the same as results using a simplified approach [8] assuming the frequency of the chirp is *constant* and equal to the initial frequency  $f_1$ :

$$\Delta\phi = 2\pi f_1 \Delta t = 2\pi f_1 \frac{2\Delta d}{c_0} = \frac{4\pi f_1 v T_c}{c_0} \quad (1.21)$$

<sup>vi</sup>Under the assumption of *complex* sampling. If real sampling was used, the sampling theorem would have the standard form of  $\frac{1}{T_p} \geq 2f_M$ .

<sup>vii</sup>Conventional values for chirp of TI AWR1642BOOST as the radar base we use in this thesis.

and so

$$v = \frac{\Delta\phi c_0}{4\pi f_1 T_c}. \quad (1.22)$$

Variable	Value
$f_1$	$77 \cdot 10^9$ Hz
$f_2$	$81 \cdot 10^9$ Hz
$T_c$	$57.14 \cdot 10^{-6}$ s
$c_0$	$3 \cdot 10^8$ m s <sup>-1</sup>

**Tab. 1.1:** Substitution values for (1.20)

The *maximum unambiguous absolute value* of velocity raises from a request on the phase difference between two adjacent chirps to lie within the  $(-\pi; \pi)$  interval. Substituting  $|\Delta\phi| < \pi$  to (1.22), we obtain

$$\left| \frac{4\pi f_1 v T_c}{c_0} \right| < \pi \quad (1.23)$$

$$|v| < \frac{c_0}{4f_1 T_c}.$$

To derive a formula for a *resolution* in velocity, recall that for a sequence of  $N$  samples taken with a sampling frequency  $f_p = 1$  Hz, the frequency gap between two adjoining samples of the corresponding DFT image is

$$\Delta f = \frac{1}{N}. \quad (1.24)$$

Using the fact and equation (1.22), we can directly write the formula for the velocity resolution of the sequence of  $N$  samples:

$$\rho_v = \frac{2\pi\Delta f c_0}{4\pi f_1 T_c} = \frac{c_0}{2Nf_1 T_c}, \quad (1.25)$$

where  $\Delta\phi = 2\pi\Delta f$ .

### 1.3.4 Range – Doppler processing

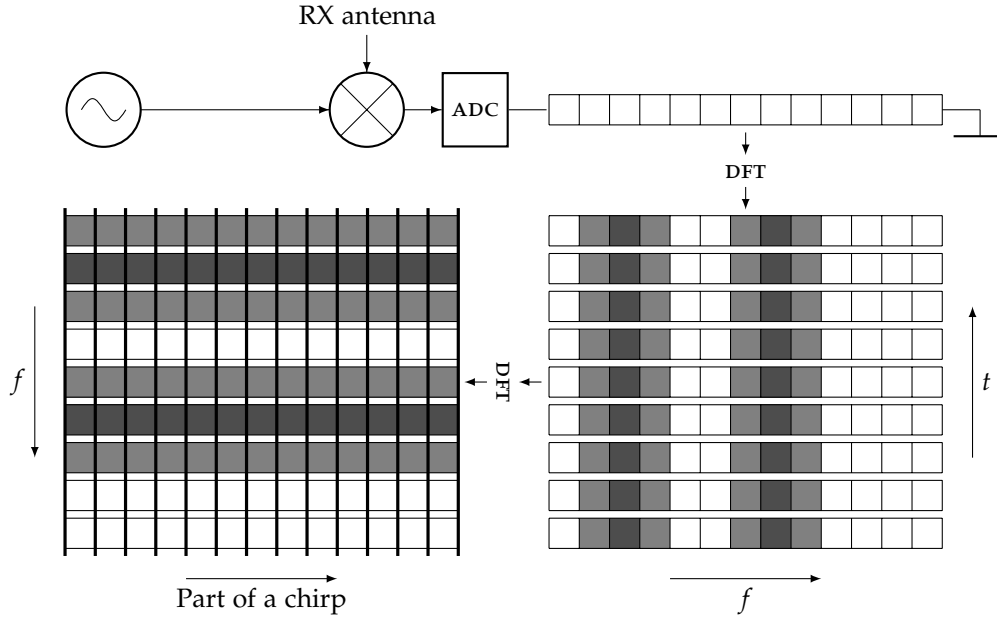
The *range – Doppler processing* is the way how to determine range and velocity of the objects in the field of view of the FMCW radar. It uses a special structure called *detection matrix* to store samples of a chirp train.

Assume that the chirp train consists of  $N_c$  chirps and for each of the chirps,  $N_s$  samples are taken. In the detection matrix, each row will contain  $N_s$  samples of one particular chirp and in each column of it, there will be samples of successive chirps aligned. Said differently, there will be sampled chirps of the complete chirp train stored in the rows of the matrix. The matrix is from the  $\mathbb{C}^{N_c \times N_s}$  class<sup>viii</sup>. Having such structure, we are able to perform DFT (discrete Fourier transforms) on

1. its rows to obtain the range information
2. columns of the matrix containing an output of the first DFT to get the velocity information

the way that is described in subsections 1.3.2 and 1.3.3. The illustrative recapitulation of the algorithm could be found in Fig. 1.7. Further, the matrix containing results of the range – Doppler processing will be called *range – Doppler matrix*.

<sup>viii</sup>Class of  $N_c \times N_s$  matrices with complex elements.



**Fig. 1.7:** The range – Doppler processing algorithm. In the figure,  $f$  stands for *frequency-related* variable as a  $x$  axis of a DFT result. The two frequency-related variables of the DFTs in the two dimensions do not generally get the same values. The range – Doppler matrix is proportional to the *sum* of the two DFT result matrices.

### 1.3.5 Multiple inputs, multiple outputs (MIMO)

In case of multiple TX or RX antennas present, we can extract even more information about the targets. Having  $N_t$  TX antennas and  $N_r$  RX antennas properly spaced, a linear array of  $N_t N_r$  antennas (so-called *virtual array*) could be simulated. Such array could be then used to extract information about the *angle* of the targets.

Assume that a reflector is far enough from the radar to ensure the same angle of arrival of the reflected signal on all the RX antennas. If there was only one TX antenna, the phase shift between two adjacent antennas can be deduced from Fig. 1.8. For the length  $\Delta d$  which the signal targeting antenna RX2 has to travel further compared to the distance to RX1 it holds

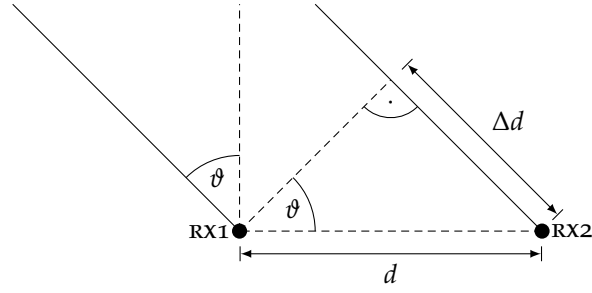
$$\Delta d = d \sin(\vartheta) \quad (1.26)$$

where  $\vartheta$  is the angle of arrival (usually denoted as AOA). The range difference  $\Delta d$  causes a phase difference between signals on RX1 and RX2

$$\Delta\phi = 2\pi f \Delta t = 2\pi f \frac{d \sin(\vartheta)}{c_0}. \quad (1.27)$$

In (1.27),  $\Delta t$  is the time difference of arrival between the two antennas. By expressing  $\vartheta$  from (1.27) we have

$$\vartheta = \arcsin\left(\frac{\Delta\phi c_0}{2\pi f d}\right). \quad (1.28)$$



**Fig. 1.8:** Two RX antennas and the incoming reflected wave.

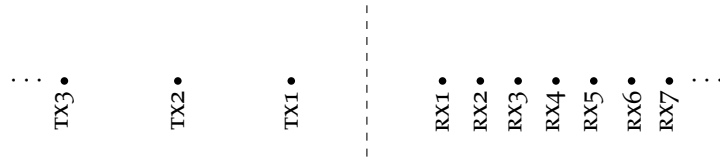
Not to allow (1.28) be ambiguous, the phase difference  $\Delta\phi$  has to be constrained to an interval “at maximum  $2\pi$  wide.” When an interval  $(-\pi; \pi)$  is selected, (1.27) yields

$$\begin{aligned} \left| \frac{2\pi f d \sin(\theta)}{c_0} \right| &< \pi \\ |\sin(\theta)| &< \frac{c_0}{2fd} \\ |\theta| &< \arcsin\left(\frac{c_0}{2fd}\right). \end{aligned} \quad (1.29)$$

Since the arcsin function constrained to  $(-\pi; \pi)$  reaches its extreme points  $\pm\pi/2$  at  $\pm 1$ , we can write an equation for maximum disambiguous range between the two antennas

$$d_M = \frac{c_0}{2f}. \quad (1.30)$$

If there were multiple TX antennas spaced so that the signal from the more distant TX antenna reaches the first RX antenna exactly  $\Delta t$  after the signal from the closer TX antenna reaches the last RX antenna, the virtual linear antenna array of length  $N_r N_t$  is simulated. Assume the indexing of TX and RX antennas as in Fig. 1.9.



**Fig. 1.9:** The indexing of TX and RX antennas.

We can then introduce the matrix of phase shifts,  $\Phi$ , which element  $\Phi_j^i$  is the phase difference between the  $i$ th TX antenna and the  $j$ th RX antenna:

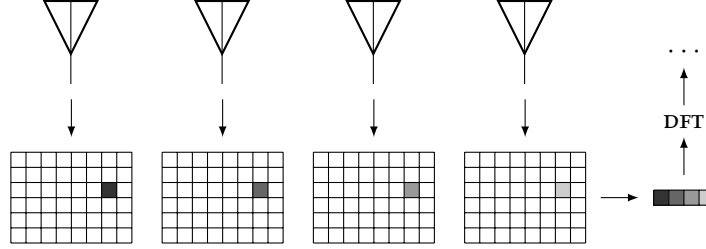
$$\Phi = [\Phi_j^i] = \begin{bmatrix} \varphi_0 & \varphi_0 + N_r \Delta\phi & \dots & \varphi_0 + (N_t - 1) N_r \Delta\phi \\ \varphi_0 + \Delta\phi & \varphi_0 + N_r \Delta\phi + \Delta\phi & \dots & \varphi_0 + (N_t - 1) N_r \Delta\phi + \Delta\phi \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_0 + (N_r - 1) \Delta\phi & \varphi_0 + N_r \Delta\phi + (N_r - 1) \Delta\phi & \dots & \varphi_0 + (N_t - 1) N_r \Delta\phi + (N_r - 1) \Delta\phi \end{bmatrix} \quad (1.31)$$

In (1.31),  $\varphi_0$  is the phase difference between TX1 and RX1.



### 1.3.6 Angle estimation

Having a linear RX antenna array, we can push the processing further and resolve an angle of arrival of the reflected signal. The principle is illustrated in Fig. 1.10.



**Fig. 1.10:** The angle estimation algorithm. The four antennas could be even *virtual*. The grids represent range – Doppler matrices the respectie antennas.

The equation (1.27) can be exploited: We can look at the phase of those bins of range – Doppler matrices for the RX antennas those all have the same range index and the same Doppler index (those which have the same “position”). The principle is based on DFT similarly with range estimation, although now the resolution of the AOA ( $\vartheta$ ) *changes with  $\vartheta$  itself*.

Assume that there are 2 objects in the field of view. The AOA of the first one is evaluated as  $\vartheta$  and that of the second one as  $\vartheta + \Delta\vartheta$ . From (1.27), the difference between the antenna-wise phase differences between the two objects ( $\Delta(\Delta\phi)$ , angular frequency) would be [9]:

$$\Delta(\Delta\phi) = \frac{2\pi fd}{c_0} (\sin(\vartheta) - \sin(\vartheta + \Delta\vartheta)) = \left\{ \cos(\vartheta) \approx \frac{\sin(\vartheta) - \sin(\vartheta + \Delta\vartheta)}{\Delta\vartheta} \right\} = \frac{2\pi fd}{c_0} \cos(\vartheta) \Delta\vartheta \quad (1.32)$$

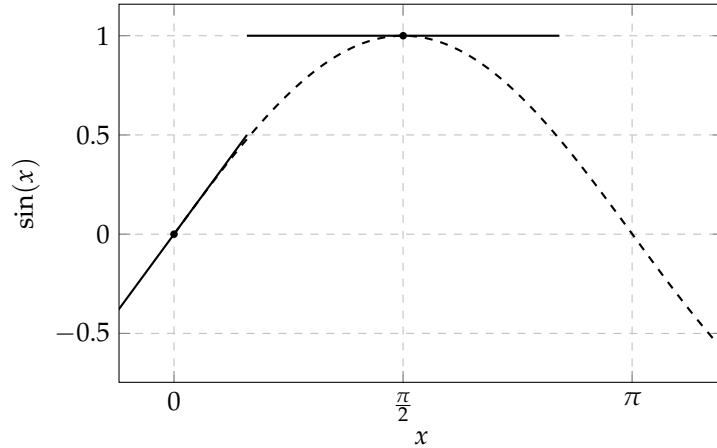
Again (see (1.24)), the frequency difference between two adjacent samples of DFT image are determined by the number of points of the input. In this case, the input is of size  $N_t N_r$ , so

$$\Delta f = \frac{1}{N_t N_r}. \quad (1.33)$$

By comparing (1.32) and (1.33) we obtain

$$\begin{aligned} 2\pi\Delta f &= \frac{2\pi fd}{c_0} \cos(\vartheta) \Delta\vartheta \\ \Delta\vartheta &= \frac{c_0}{N_t N_r f d \cos(\vartheta)}. \end{aligned} \quad (1.34)$$

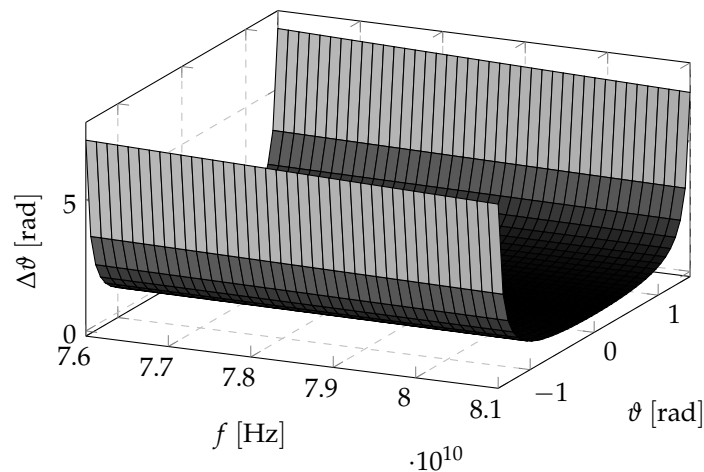
The fact that the angle resolution depends on the AOA is easy to be admitted as true: When the AOA goes to  $\pi/2$  rad, the cosine term of the denominator of (1.34) approaches zero and the angle resolution becomes infinite. The dependence of  $\Delta\phi$  on  $\vartheta$  will be infinitely weak since the dependence of  $\sin(\vartheta)$  on  $\vartheta$  is infinitely small. The tangent of the sine function in  $\pi/2$  has zero slope. Oppositely, the angle resolution is maximum when the AOA is zero; That means when the target is close in angle to the direction of the main beam. The tangents of sine function in 0 rad and  $\pi/2$  rad is shown in Fig. 1.11.



**Fig. 1.11:** The sine function and its tangents in 0 rad and  $\pi/2$  rad.

To recapitulate the angle resolution problem, we can generally say that the more virtual antennas the better resolution, and also the more is the beam focused on the target, the better the resolution is.

Let us address the question of a spacing between the rx antennas. Although this chapter tries to be as *general* as possible, in this case it is appropriate to be device-specific. In Chapter 2, we will find out that the TI AWR1642BOOST radar kit is employed. In the kit's reference manual [10], the rx antennas spacing is declared to be  $\lambda/2$ . That could sound a bit insufficient information since frequency of the radar signal (linear chirp) is not constant and therefore the wavelength  $\lambda$  is not constant. However, if we took into account the maximum allowed bandwidth of the chirp (from 76 GHz to 81 GHz), we come across wavelengths from 3.7 mm to 3.95 mm. The maximum difference between the values of  $\lambda/2$  connected with two distinct frequencies within this band is approximately 0.128 mm which does not make a significant difference. The angle resolution as a function of the AOA and chirp frequency is depicted in Fig. 1.12.

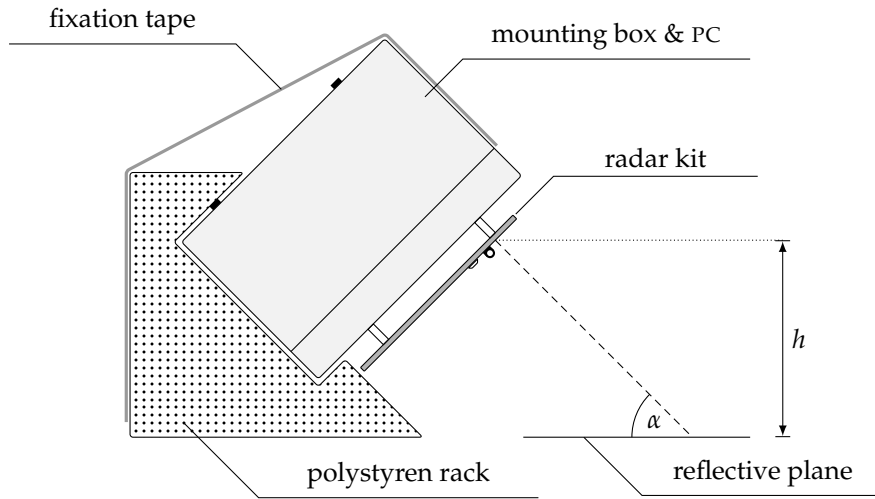


**Fig. 1.12:** The angle resolution as a function of the AOA and chirp frequency.

## 1.4 Radar sensor geometry

The radar and vehicle setup that will be used over this thesis is the one roughly described in the end of section 1.2.1. The monostatic radar is placed on the top of a vehicle and targets its main beam to the ground plane under a depression angle  $\alpha$ . Such setup is straightforward since the ground plane

is the one and only reflector that is *always* present for almost all of the ground vehicles and which velocity towards the radar device, fixed on the vehicle, directly reflects the vehicle's velocity. The final setup is sketched in Fig. 1.13. The particular components marked in the picture will be explained later in Chapter 2.



**Fig. 1.13:** The radar geometry used in this thesis.

## Hardware and its limitations

In order to test signal processing algorithms we need a real radar hardware (HW). The radar has to obey some general rules in order to be useful/applicable in real world. The rules rise from two essential factors: the law and price. Based on a particular radar design employed, appropriate processing HW and auxiliary HW, e.g. power supply have to be chosen. In this chapter, the particular HW elements of the prototype will be addressed.

### 2.1 Market options

Because an experimentally tested and successful velocity evaluation algorithm has been designed within our past research [11], there was an effort to take an advantage from it. That, however, requested the CW radar system to be used, which turned out to be a problem. On the market, there was none of the out-of-the-box CW systems and construction of the custom one from scratch appeared as a serious research problem itself [12]. A 77 GHz CW radar architecture was designed and simulated in [13], however, our goal was to end up with a working, real, possibly plug-and-play device. Early we realized that we have to adapt to the market possibilities and move our focus to the group of FMCW devices available.

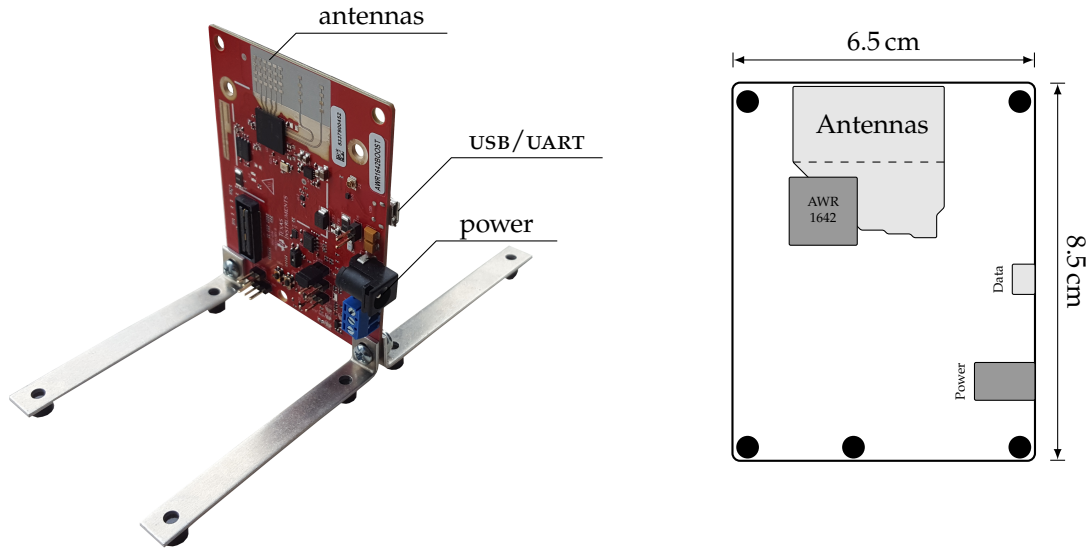
The first possibility is the MR3003\_RD reference design board from Swiss company RFbeam Microwave [14]. It offers FMCW module with frequency range from 76 GHz to 81 GHz, control panel, or an ethernet interface. Nevertheless, the device is rather expensive (total price including software support approximately 3700 €). It is based on S32R274 radar chip from NXP.

Dutch company NXP offers its own reference design kits based on the S32R274 chip [15]. They offer the kit in a plastic box together with a graphical user interface (GUI), schematics, or Gerber files and technical video support (depends on the version of the kit – standard or precise). The price tag is in the \$3500 range. Both of the previous systems require an external power source.

The third possible solution is presented by Texas Instruments (TI). They offer radar evaluation boards based on their AWR1642 or AWR1443 radar sensors. There are multiple boards in their portfolio, those vary in the particular sensor used and in the connectivity options. They all contain ARM controllers. As the only manufacturer they offer a 76 GHz to 81 GHz FMCW evaluation module (EVM) at price below 300 USD.

### 2.2 The radar board

Since we wanted to measure horizontal velocity of vehicles moving over the ground, the TI AWR1642BOOST module has been purchased. When compared to the similar IWR1642BOOST, it contains the CAN bus interface that could be usable later if the speedometer we have designed was to be deployed directly in a car. The TI AWR1642BOOST kit is in Fig. 2.1(a) and its dimensions are given in Fig. 2.1(b).



(a) The TI AWR1642BOOST evaluation module.

(b) Dimensions of the TI AWR1642BOOST radar EVM. The mounting holes are compatible with M3 screws.

**Fig. 2.1:** Appearance and dimensions of the radar EVM.

The board arrived with an USB- $\mu$ USB cable and some screws, nuts and rack parts to put the EVM in vertical position (see Fig. 2.1(a)). Neither the power source nor the cable with 2.1 mm barrel jack was included.

### 2.2.1 Components of the board

The board itself contains various features, most of which we will not even use. Those which are important for our application are:

**Radar sensor.** A functional block diagram of the radio subsystem of the TI AWR1642 sensor is in Fig. 2.2. Besides the radio subsystem, the sensor integrates two processors:

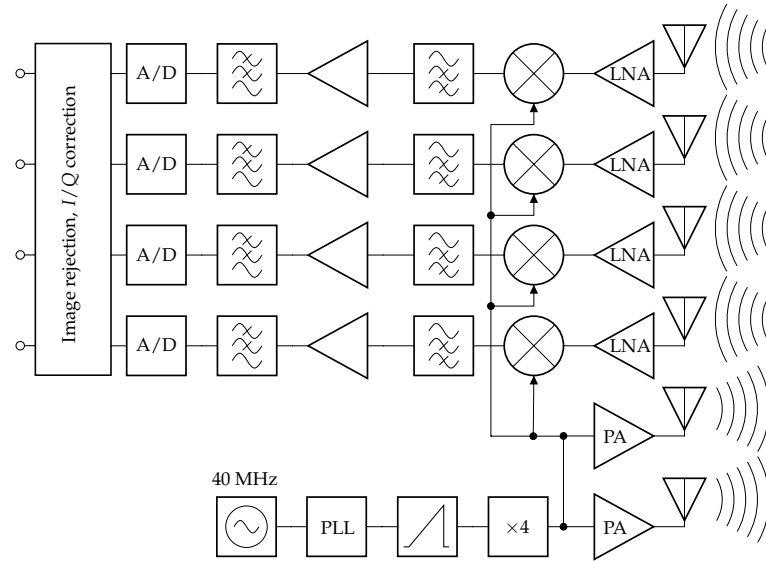
1. ARM Cortex R4F with 200 MHz clock (master subsystem)
2. TI C674X DSP (digital signal processor) with 600 MHz clock frequency (DSP subsystem).

All the useful information about the chip is summarized in [16].

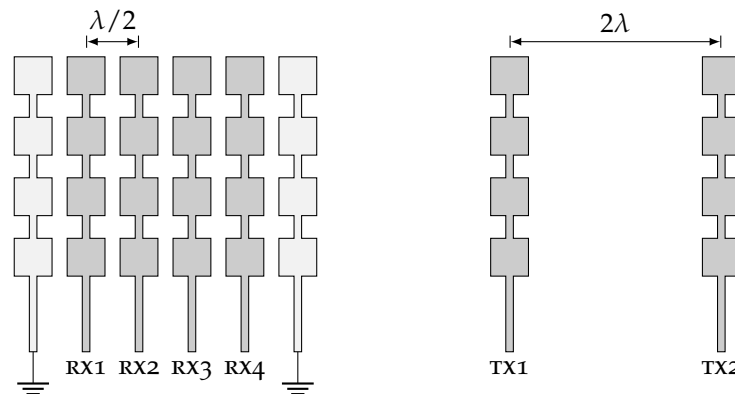
**Antennas.** The EVM disposes with 4 RX and 2 TX patch antenna arrays (further only *antennas*). In addition, there are two inactive antennas those probably act like dummy elements, equalizing the mutual coupling for all the active RX antennas [17]. The spacing between the RX antennas is 1.91 mm, which corresponds to half a wavelength (the “middle one”, tied with frequency 78.5 GHz), and the spacing between the two TX antennas is  $2\lambda$  at the same frequency. See section 1.3.5 for reasoning. The sketch of the antenna shape and positioning is in Fig. 2.3. More about the antennas and their parameters could be found right in the next subsections.

**Micro USB connector** with a XDS110 emulator, providing JTAG debugger and UART [10].

**Mode-determining jumpers.** There are three jumpers on the board. These determine an operation mode of the EVM. The jumpers are named SOP0, SOP1, and SOP2, where SOP means *sense-on-power*. The three modes and the respective positions of the jumpers are summarized in Tab. 2.1 [18].



**Fig. 2.2:** The functional block diagram of the RF subsystem of the radar sensor TI AWR1642. The ramp generator block encapsulates a timing engine and a frequency synthesizer with output frequency  $\approx 20$  GHz. The antennas are not included in the sensor itself. All the processing in the RX part is done in *quadrature* manner (not explicitly drawn in the diagram). From [16], edited.





**Fig. 2.3:** The antennas of the AWR1642BOOST radar EVM. The two antennas of the different color are the dummy elements. From [10], edited.

Moreover, there are interfaces for various buses, a power consumption measurement interface, reset button, LEDs, etc.

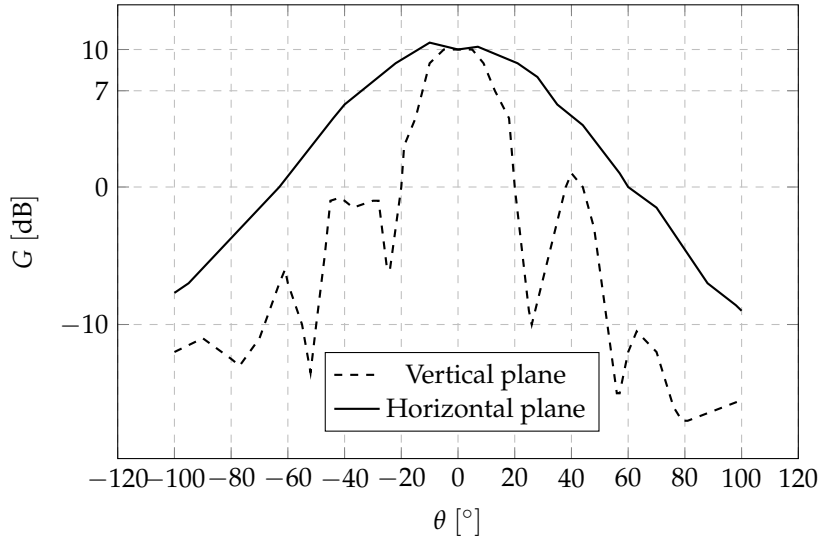
### 2.2.2 Antenna pattern

An antenna<sup>i</sup> power pattern for frequency 78.5 GHz is available in [10] in the form of picture (a plot). The part close to the main lobe is replotted in 2.4 from deduced data. The fundamental characteristics derived from the power pattern are in Tab. 2.2.

<sup>i</sup>Only *one* antenna array of the 6 active of them present on the board is addressed. One antenna array means one “column” in Fig. 2.3.

Mode	Jumper positions (SOP2, 1, 0)	Usage
Flash programming		Flashing an application binary
Functional		Loading an application from the serial flash to internal RAM

**Tab. 2.1:** The two used operational modes of the EVM.



**Fig. 2.4:** The power pattern of the elementary antenna array (one column of RX or TX array).

3 dB supression		10 dB supression		Full main beam	
Vert.	Hor.	Vert.	Hor.	Vert.	Hor.
0.489 rad	1.152 rad	0.698 rad	2.094 rad	1.047 rad	3.491 rad

**Tab. 2.2:** Beamwidths of one AWR1642BOOST antenna in vertical (Vert.) and horizontal (Hor.) planes for different suppressions.

### 2.2.3 Far field

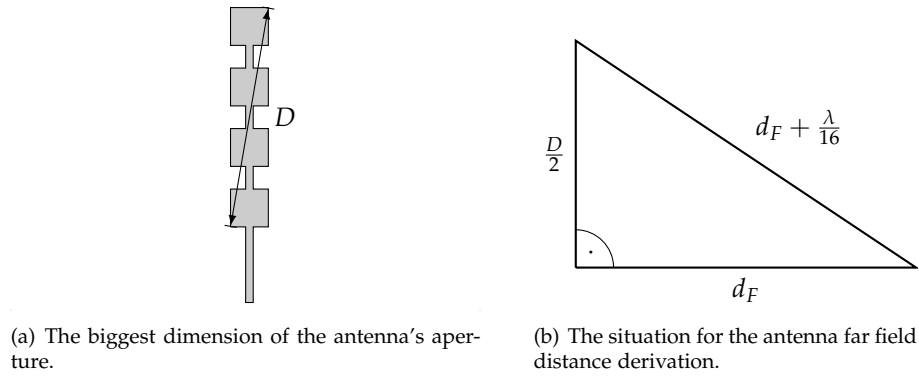
Far field distance in terms of maximum phase difference is defined as the distance  $d_F$ , at which the maximum phase difference between the real wave and its ideal plane wave approximation within the circle area symmetric about the main antenna beam axis and perpendicular to it, with diameter  $D$  is  $\Delta\phi_F$ . Hereby  $D$  is the largest dimension of the antenna's aperture, and the standard value of  $\Delta\phi_F$  is  $\pi/8$  [19]. The problem of determining  $d_F$  leads to a simple geometric excersise given by 2.5(b).

Solution leads to the well-known formula

$$d_F = \frac{2D^2}{\lambda} = \frac{2D^2 f}{c_0} \tag{2.1}$$

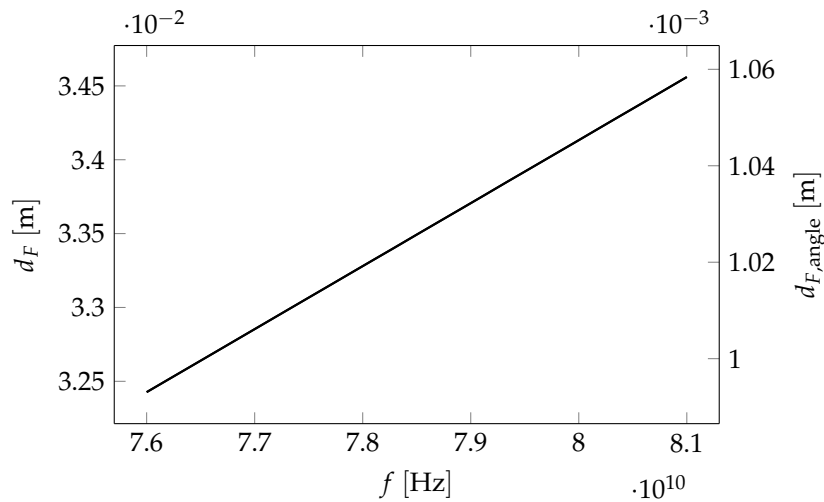
where  $c_0$  is velocity of light in vacuum as the maximum possible velocity of EM wave propagation. In case of TI AWR1642BOOST antenna, the distance of far field results to

$$d_F = \frac{2 \cdot (8 \cdot 10^{-3})^2 \cdot 81 \cdot 10^9}{3 \cdot 10^8} = 3.456 \text{ cm} \tag{2.2}$$



**Fig. 2.5:** On a derivation of the antenna far field distance.

since the highest useful frequency that could be present in the chirp signal of the EVM is 81 GHz [16]. The highest frequency value, however, depends on the particular chirp used – the value from (2.2) serves only as an upper boundary. Dependence of  $d_F$  on the maximum chirp frequency is plotted in Fig. 2.6. It is worth pointing out here that even though the four RX antennas look like an antenna array, they work independently in case of *range*, and *velocity* estimation. Therefore, the biggest antenna dimension  $D$  is the biggest dimension of *one particular antenna*<sup>ii</sup>. For angle estimation, we put a request of being planar on the wave impinging the whole RX antenna array. In such case, the biggest dimension of the receiving antenna is approximately 14 mm (the diagonal of the RX array in Fig. 2.3, including the dummy elements), and the far field distance results in  $d_{F,\text{angle}} = 10.584$  cm.



**Fig. 2.6:** The far field distance dependence on the highest chirp frequency for the elementary antenna array, and the complete RX array of the TI AWR1642Boost (values for  $\Delta\phi_F = \pi/8$ ).

## 2.2.4 Area of reflection

When a radar transmits a general wave with its main beam directed to the ground (see Fig. 1.13), the *significant* part of the wave reflects back to the radar from an area that we call an *area of reflection* (AOR) within this thesis. The precise knowledge of such area's shape allows us to design the positioning of the radar on the vehicle properly. On the following lines, the shape of the AOR will be derived.

<sup>ii</sup>One antenna is still an array of four elementary patch antennas.

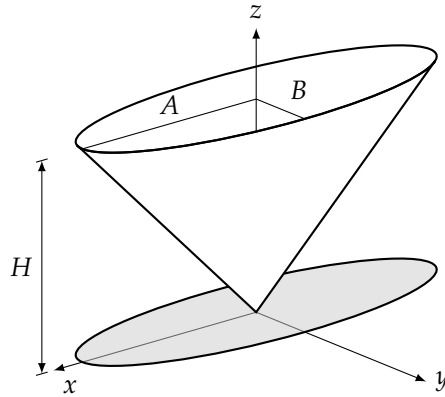


Firstly, assume that we are only concerned with that part of the wave that lies within the space angle with less than 3 dB loss against the maximum directivity of the antenna: We expect that, over the ideal flat ground all the reflections from outside of such area are too weak to make a significant contribution to the reflected signal. Unfortunately, in the real case there could be a reflector with better reflection than any of the ground elements of the AOR present outside the AOR. That case, however, does not fall into the purely geometrical problem of this subsection.

According to 2.2.2, we know the 3 dB beamwidths of the examined antenna in both the vertical and the horizontal planes ( $\theta_v$  and  $\theta_h$ , respectively):

$$\theta_v = 0.489 \text{ rad} \quad \text{and} \quad \theta_h = 1.152 \text{ rad}. \quad (2.3)$$

From that, the 3 dB beam (named only *beam* further within this subsection) has the shape of an elliptic cone. Placed into cartesian coordinate system, the cone is sketched in Fig. 2.7.



**Fig. 2.7:** The considered placement and positioning of the elliptic cone representing the 3 dB beam of the radar antenna in cartesian coordinates  $(x, y, z)$ .

Our goal is to find an area formed by an intersection of the elliptic cone from Fig. 2.7, and a plane representing the ground. In Fig. 2.7,  $H$  is the distance between the vertex of the cone and the intersect point of the  $z$  axis and the ground plane.  $A$  and  $B$  are the semi-major and the semi-minor axis of the cone in the plane parallel to the  $xy$  plane and elevated by  $H$ , respectively. The equation of the cone from Fig. 2.7 is [20]:

$$\frac{x^2}{A^2} + \frac{y^2}{B^2} = \frac{z^2}{H^2}. \quad (2.4)$$

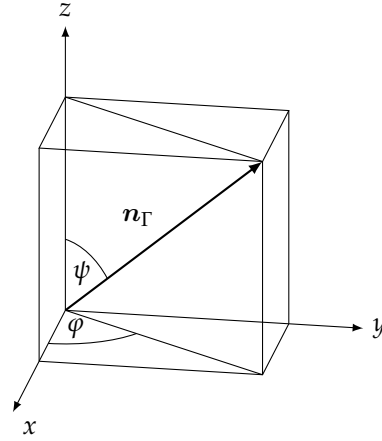
The intersection of the cone and the ground plane<sup>iii</sup> lies in the ground plane, admittedly. In Fig. 2.7, the ground plane is not drawn, however, it can be described by any pair of vectors in  $xyz$  space which are not colinear. If we rotate the ground plane the way its normal is colinear with the  $z$  axis, we are able to fully describe the intersection using only two— $x'$  and  $y'$ —coordinates of the  $(x', y', z')$  reference frame, where the  $(x', y', z')$  frame is the  $(x, y, z)$  frame translated along its  $z$  axis [21].

Let the ground plane be denoted as  $\Gamma$ . Let us find the matrix of rotation of  $\Gamma$ , that, applied to any vector of  $\Gamma$ , outputs a vector parallel to the  $xy$  plane. The problem is equivalent to a problem of rotating the normal vector  $\mathbf{n}_\Gamma$  of  $\Gamma$  to the direction of the  $z$  axis. The situation is drawn in Fig. 2.8.

To make  $\mathbf{n}_\Gamma$  colinear with the  $z$  axis we need to (referring to Fig. 2.8):

1. Rotate the  $(x, y, z)$  frame by the angle  $\varphi$  around the  $z$  axis so we obtain a new reference frame  $(\bar{x}, \bar{y}, z)$ .
2. Rotate the new  $(\bar{x}, \bar{y}, z)$  frame around its  $\bar{y}$  axis by the angle  $-\psi$ .

<sup>iii</sup>We may refer to it as just *the intersection*.

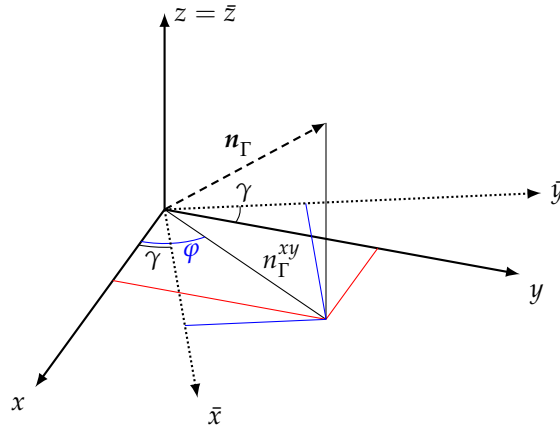


**Fig. 2.8:** The normal vector  $n_\Gamma$  of the general ground plane in the  $(x, y, z)$  reference frame of Fig. 2.7.

If we denoted the matrix of rotation around the general  $z$  axis by angle  $\gamma$  as  $R_z^\gamma$  and, similarly, the matrix of rotation around the general  $y$  axis by angle  $\beta$  as  $R_y^\beta$ , we could achieve the desired transformation as

$$n'_\Gamma = R_y^{-\psi}(R_z^\varphi n_\Gamma) = R_y^{-\psi} R_z^\varphi n_\Gamma = R n_\Gamma. \quad (2.5)$$

To derive the general form of  $R_z^\gamma$  we will use Fig. 2.9.



**Fig. 2.9:** On rotation of the  $(x, y, z)$  frame about the  $z$  axis.  $n_\Gamma^{xy}$  is a projection of the vector  $n_\Gamma$  to the  $xy$  plane.

It is clear that the  $z$  axis will not be influenced by the rotation:

$$\bar{z} = z. \quad (2.6)$$

For  $\bar{x}$  and  $\bar{y}$  we can write

$$\begin{aligned} \bar{x} &= n_\Gamma^{xy} \cos(\varphi - \gamma) = \underbrace{n_\Gamma^{xy} \cos(\varphi)}_x \cos(\gamma) + \underbrace{n_\Gamma^{xy} \sin(\varphi)}_y \sin(\gamma) \\ \bar{y} &= n_\Gamma^{xy} \sin(\varphi - \gamma) = \underbrace{n_\Gamma^{xy} \sin(\varphi)}_x \cos(\gamma) - \underbrace{n_\Gamma^{xy} \cos(\varphi)}_y \sin(\gamma), \end{aligned} \quad (2.7)$$

so from (2.6) and (2.7) by rewriting to the matrix form

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}_z^\gamma} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (2.8)$$

we eventually obtain

$$\mathbf{R}_z^\gamma = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.9)$$

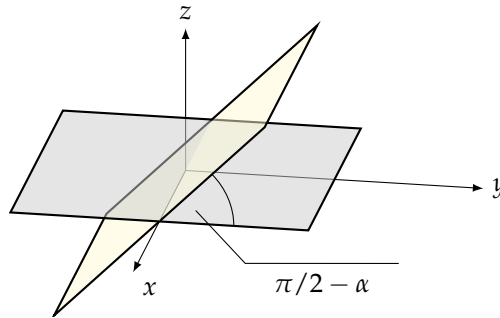
The general form of  $\mathbf{R}_y^\beta$  could be derived similarly, resulting to

$$\mathbf{R}_y^\beta = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix}. \quad (2.10)$$

Substituting  $\gamma = \varphi$ ,  $\beta = -\psi$  in (2.9) and (2.10) and multiplying the two we obtain the overall rotation matrix

$$\mathbf{R} = \begin{bmatrix} \cos(\varphi)\cos(\psi) & \sin(\varphi)\cos(\psi) & \sin(\psi) \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ -\cos(\varphi)\sin(\psi) & -\sin(\varphi)\sin(\psi) & \cos(\psi) \end{bmatrix}. \quad (2.11)$$

Nevertheless, the situation we are concerned with is slightly less general. Since the radar antenna only has a depression angle  $\alpha$  and an azimuth and a tilt are both zero, it is possible to draw the ground plane to the  $(x, y, z)$  coordinate system as in Fig. 2.10.



**Fig. 2.10:** The positioning of the ground plane in the  $(x, y, z)$  coordinate frame in the desired radar setup.

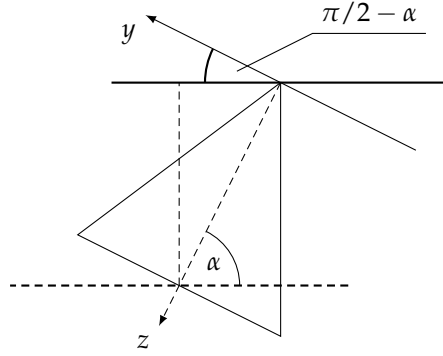
In Fig. 2.10 the background behind the angle  $\pi/2 - \alpha$  could be a source of confusion. Probably, it would be more clear from Fig. 2.11 – a 2D cut of the situation in Fig. 2.10 rotated such that it resembles the suggested radar-on-the-vehicle setup.

The overall rotation matrix  $\mathbf{R}$  therefore reduces to the matrix of rotation about the  $x$  axis by the angle  $(\pi/2 - \alpha)$ . Substituting  $\alpha' = \pi/2 - \alpha$  in the general form (derived in analogy with (2.8))

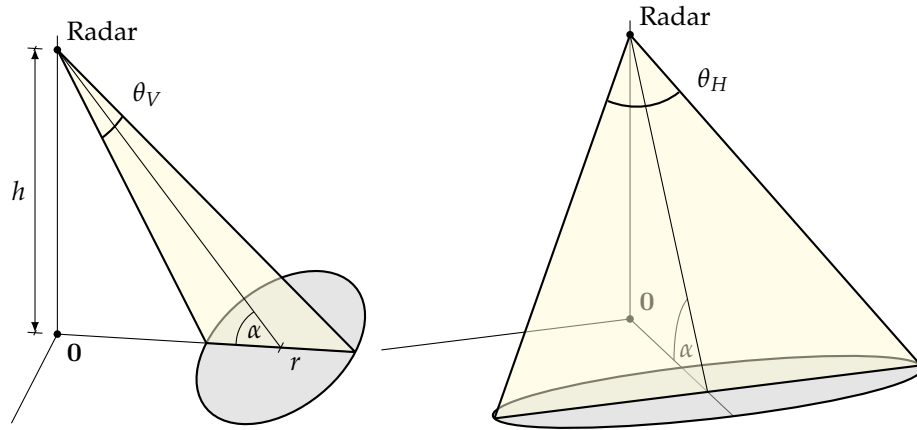
$$\mathbf{R}_x^{\alpha'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha') & \sin(\alpha') \\ 0 & -\sin(\alpha') & \cos(\alpha') \end{bmatrix} \quad (2.12)$$

we get

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) \\ 0 & -\cos(\alpha) & \sin(\alpha) \end{bmatrix}. \quad (2.13)$$



**Fig. 2.11:** 2D picture of the situation in Fig. 2.10. The radar is placed in the vertex of the cone.



**Fig. 2.12:** Theoretical radar deployment setup. The same situation from the field of view of two different perspectives.

Now, let us express the variables  $A$ ,  $B$  and  $H$  of (2.4) in terms of the real placement of the radar on the vehicle. Consider the situation in Fig. 2.12.

In Fig. 2.12, the yellow triangles are cuts of the radar beam cut by the ground plane. The gray shapes represent exactly the intersection which boundary we effort to describe. Comparing Fig. 2.7 and Fig. 2.12 we can write

$$\begin{aligned} H &= \frac{h}{\sin(\alpha)} \\ A &= \frac{h \tan(\theta_h/2)}{\sin(\alpha)} \\ B &= \frac{h \sin(\theta_v/2)}{\sin(\alpha) \sin(\pi - \alpha - \theta_v/2)}. \end{aligned} \quad (2.14)$$

Now, let the ground plane be described by the point-normal equation in the  $(x, y, z)$  reference frame as

$$\mathbf{n}_\Gamma^T \mathbf{r} + d = 0, \quad (2.15)$$

where  $\mathbf{n}_\Gamma = [a, b, c]^T$  is its normal,  $\mathbf{r} = [x, y, z]^T$  is an arbitrary vector lying in  $\Gamma$  and  $d$  is the distance between (possibly translated) origin of  $\mathbf{n}_\Gamma$  and the origin of the  $(x, y, z)$  frame. Note that we take into account orientation of vectors (row vs. column) to become able to identify matrix product with euclidean scalar product. If we denote the  $\mathbf{r}$  vector expressed in  $(x', y', z')$  reference frame as

$\mathbf{r}' = [x', y', z']^T$ , considering the properties of  $\mathbf{R}$  we can write

$$\mathbf{r}' = \mathbf{R}\mathbf{r} \quad (2.16)$$

and because of orthogonality<sup>iv</sup> of rotation matrix  $\mathbf{R}$ , (2.16) yields

$$\mathbf{r} = \mathbf{R}^{-1}\mathbf{r}' = \mathbf{R}^T\mathbf{r}'. \quad (2.17)$$

Matrix  $\mathbf{R}$  was designed such that

$$\mathbf{R}\mathbf{n}_\Gamma = [0, 0, |\mathbf{n}_\Gamma|]. \quad (2.18)$$

Taking equations (2.15), (2.17) and (2.18) into account, we have

$$\begin{aligned} \mathbf{n}_\Gamma^T \mathbf{R}^{-1}\mathbf{r}' + d &= 0 \\ \mathbf{n}_\Gamma^T \mathbf{R}^T\mathbf{r}' + d &= 0 \\ (\mathbf{R}\mathbf{n}_\Gamma)^T \mathbf{r}' + d &= 0 \\ [0, 0, |\mathbf{n}_\Gamma|][x', y', z']^T + d &= 0 \\ z' &= -d/|\mathbf{n}_\Gamma|. \end{aligned} \quad (2.19)$$

Let us return back to the equation of an elliptic cone (2.4). Rewritten in terms of an arbitrary vector  $\mathbf{r}$  it is

$$\mathbf{r}^T \underbrace{\begin{pmatrix} 1/A^2 & 0 & 0 \\ 0 & 1/B^2 & 0 \\ 0 & 0 & -1/H^2 \end{pmatrix}}_C \mathbf{r} = 0. \quad (2.20)$$

We can modify (2.20) using the previous relations and obtain subsequently

$$\begin{aligned} (\mathbf{R}^{-1}\mathbf{r}')^T C \mathbf{R}^{-1}\mathbf{r}' &= 0 \\ (\mathbf{R}^{-1}\mathbf{r}')^T C \mathbf{R}^T\mathbf{r}' &= 0 \\ \mathbf{r}'^T \underbrace{C \mathbf{R}^T}_M \mathbf{r}' &= 0 \end{aligned} \quad (2.21)$$

$$[x', y', -d/|\mathbf{n}_\Gamma|] \mathbf{M} [x', y', -d/|\mathbf{n}_\Gamma|]^T = 0.$$

In the last line of (2.21), there is more or less the equation we aimed to – in the “elevated”  $xy$  plane— $x'y'$  plane—it describes the intersection of the 3 dB radar beam and the ground plane in variables  $x'$  and  $y'$ . We can further expand (2.21) using

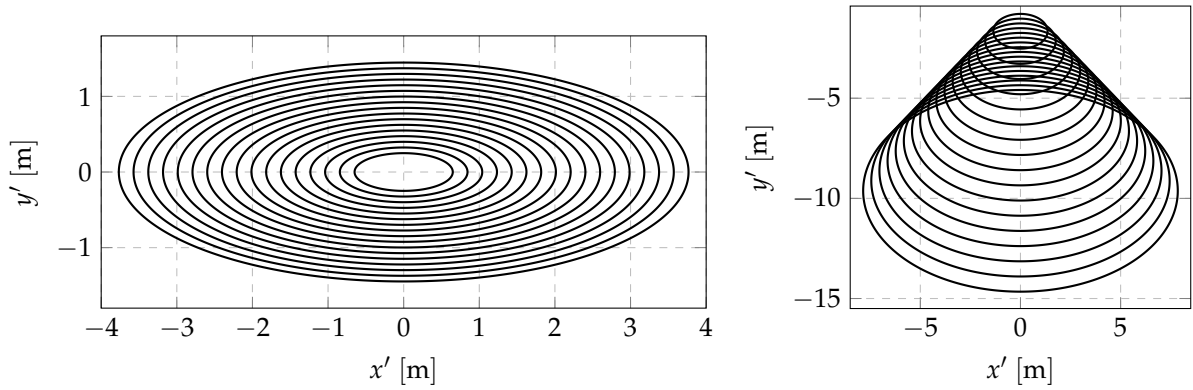
$$\mathbf{M} = \begin{pmatrix} \frac{\sin^2(\alpha)}{h^2 \tan^2(\frac{\theta_p}{2})} & 0 & 0 \\ 0 & \frac{\sin^4(\alpha) \sin^2(\alpha + \frac{\theta_p}{2} - \pi)}{h^2 \sin^2(\frac{\theta_p}{2})} - \frac{\sin^2(\alpha) \cos^2(\alpha)}{h^2} & 0 \\ 0 & -\frac{\sin^3(\alpha) \cos(\alpha)}{h^2} - \frac{\sin^3(\alpha) \sin^2(\alpha + \frac{\theta_p}{2} - \pi) \cos(\alpha)}{h^2 \sin^2(\frac{\theta_p}{2})} & 0 \\ & & -\frac{\sin^3(\alpha) \cos(\alpha)}{h^2} - \frac{\sin^3(\alpha) \sin^2(\alpha + \frac{\theta_p}{2} - \pi) \cos(\alpha)}{h^2 \sin^2(\frac{\theta_p}{2})} \\ & & -\frac{\sin^4(\alpha)}{h^2} + \frac{\sin^2(\alpha) \sin^2(\alpha + \frac{\theta_p}{2} - \pi) \cos^2(\alpha)}{h^2 \sin^2(\frac{\theta_p}{2})} \end{pmatrix}. \quad (2.22)$$

The final equation of the intersection is then

---

<sup>iv</sup>  $\mathbf{R}^T = \mathbf{R}^{-1}$ .

$$\begin{aligned}
& h \left( \frac{h \left( -\frac{\sin^4(\alpha)}{h^2} + \frac{\sin^2(\alpha) \sin^2(\alpha + \frac{\theta_v}{2} - \pi) \cos^2(\alpha)}{h^2 \sin^2(\frac{\theta_v}{2})} \right)}{\sqrt{|\sin(\alpha - \frac{\pi}{2})|^2 + |\cos(\alpha - \frac{\pi}{2})|^2 \sin(\alpha)}} + y' \left( -\frac{\sin^3(\alpha) \cos(\alpha)}{h^2} - \frac{\sin^3(\alpha) \sin^2(\alpha + \frac{\theta_v}{2} - \pi) \cos(\alpha)}{h^2 \sin^2(\frac{\theta_v}{2})} \right) \right) \\
& - \frac{\sqrt{|\sin(\alpha - \frac{\pi}{2})|^2 + |\cos(\alpha - \frac{\pi}{2})|^2 \sin(\alpha)}}{\sqrt{|\sin(\alpha - \frac{\pi}{2})|^2 + |\cos(\alpha - \frac{\pi}{2})|^2 \sin(\alpha)}} + \\
& + y' \left( -\frac{h \left( -\frac{\sin^3(\alpha) \cos(\alpha)}{h^2} - \frac{\sin^3(\alpha) \sin^2(\alpha + \frac{\theta_v}{2} - \pi) \cos(\alpha)}{h^2 \sin^2(\frac{\theta_v}{2})} \right)}{\sqrt{|\sin(\alpha - \frac{\pi}{2})|^2 + |\cos(\alpha - \frac{\pi}{2})|^2 \sin(\alpha)}} + \right. \\
& \left. + y' \left( \frac{\sin^4(\alpha) \sin^2(\alpha + \frac{\theta_v}{2} - \pi)}{h^2 \sin^2(\frac{\theta_v}{2})} - \frac{\sin^2(\alpha) \cos^2(\alpha)}{h^2} \right) \right) + \frac{x'^2 \sin^2(\alpha)}{h^2 \tan^2(\frac{\theta_h}{2})} = 0 \quad (2.23)
\end{aligned}$$



(a)  $\alpha = 90^\circ$ . Expectedly, the shapes share the same center and get bigger with increasing  $h$ . (b)  $\alpha = 45^\circ$ . Expectedly, the shapes move their centers in linear manner with increasing  $h$ .

**Fig. 2.13:** The intersections of the radar cone and the ground plane for two values of  $\alpha$  ( $\alpha = 45^\circ$ ,  $\alpha = 90^\circ$ ), and  $h$  varying from 1 m to 6 m with step 0.3 m.

Equation (2.23) could be used to compute the *shape* of the intersection. Finally, let us confirm the correctness of the result by drawing a few intersection plots: Fig. 2.13(a), Fig. 2.13(b) and Fig. 2.14. For all of them, the radar is placed in  $[0, 0]^T$ .

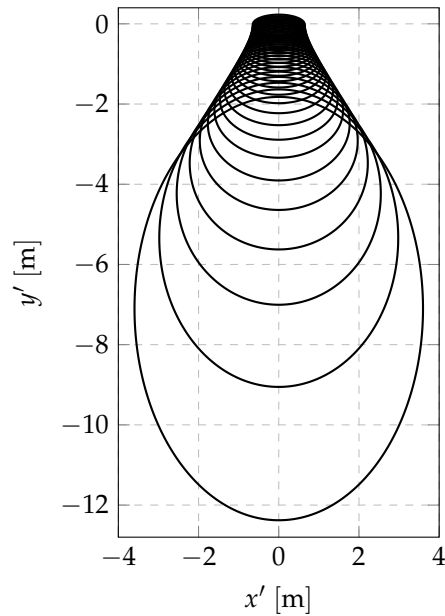
## 2.3 Computer subsystem

### 2.3.1 Raw ADC samples

The radar EVM itself does not output raw ADC samples. The reason is that the bitrate of the UART as the data output interface is too low to serve for all the ADC samples. There is, however, a possibility for getting them provided directly by the manufacturer. The raw samples are stored in so-called ADC buffer before they are moved to the special section of memory (L3 memory) [22]. The manufacturer offers the DCA1000EVM – the FPGA-based device able to capture the samples from the memory and send raw ADC samples via an Ethernet interface. Even if it could be fairly advantageous to have an access to the raw ADC samples, the price of the capture card (\$499<sup>v</sup>) has been beyond our budget.

Another approach to the raw ADC samples is proposed in [23]. According to [16, 22], there is a 768 kB L3 memory shared between the main processor and the DSP. This memory serves as a data

<sup>v</sup>The price could change indeed.



**Fig. 2.14:** The intersections of the radar cone and the ground plane for  $h = 1$  m and  $\alpha$  varying from  $90^\circ$  to  $29^\circ$  with step  $2^\circ$ .

store for one frame of chirps<sup>vi</sup>. By accessing the L3 memory, it is possible to read the values of raw samples for one frame.

To do that, the CCS development mode of the TI AWR1642BOOST has to be used. The mode is based on flashing a special program to the EVM and running the program under test virtually, using the CCS software development tool. The CCS uses the XDS110 emulator to provide an access to the memories of the EVM and to handle the data flow and functions of the program under test. In this mode, an user is able to send and view the content of the EVM's memory.

For an implementation, the *mmWave demo* program distributed by the TI AWR1642BOOST has been used. The program provides for the cooperation of the subsystems of the EVM, supervises the signal processing and could be controlled via the integrated CLI (command line interface). A new CLI command has been added. The command is called `printAddressContent` and its establishment rests in three small pieces of code added to the `cli.c` source of the *mss* part of the *mmWave demo*:

1. Declaration of the callback function at the section *Local definitions* (line cca. 95)

```
1 static int32_t MmwDemo_CLIprintAddressContent (int32_t argc, char* argv[]);
```

**Listing 2.1:** Declaration of the callback function.

2. Definition of the function itself

```
1 static int32_t MmwDemo_CLIprintAddressContent (int32_t argc, char * argv[]) {
2     bool doReconfig = false;
3     CLI_write("%08lx\n", *((int32_t *)0x51020000));
4     return(0);
5 }
```

**Listing 2.2:** Definition of the callback function for the `printAddressContent` command.

3. Adding the new CLI command to the table of `MmwDemo_CLIIInit` function (line cca. 1300)

<sup>vi</sup>For definition of a frame see Sec. 3.1.2.

```

1 cliCfg.tableEntry[cnt].cmd      = "printAddressContent";
2 cliCfg.tableEntry[cnt].helpString = "No_arguments";
3 cliCfg.tableEntry[cnt].cmdHandlerFxn = MmwDemo_CLIprintAddressContent;
4 cnt++;

```

**Listing 2.3:** Adding the new CLI command to the command table.

In Lst. 2.2 the address  $51020000_{16}$  is the start address of the L3 memory block. The address has been retrieved empirically, by observation of changes in memory content using the CCS. The stored samples are of `int32_t` data type according to [22].

The command has been modified so that it reads not only one, but *all* the raw ADC samples of the actual chirp train. The callback definition has been changed to run in the loop and print several consecutive values:

```

1 static int32_t MmwDemo_CLIprintAddressContent (int32_t argc, char * argv[]) {
2     int32_t i;
3     const int32_t N_SAMPLES_2B_PRINTED = 10;
4
5     bool doReconfig = false;
6     for (i = 0; i < N_SAMPLES_2B_PRINTED; i++) {
7         CLI_write("%08lx\n", *((int32_t *) (0x51020000 + i * 4)));
8     }
9     return(0);
10 }

```

**Listing 2.4:** The modification of Lst. 2.2 for printing several consecutive values from the L3 memory.

Nevertheless, the approach has not bring sufficient results. There are few problems with it, particularly:

1. Each of the CLI commands has a *timeout* set within the main program. The timing of the processing is crucial in order to achieve stable operation. On one hand, we would need to extend the timeout to be able to send as many samples as we need; On the other hand, touching the timeout value influences other parts of the processing and the program does not behave as expected.
2. It is not possible to retrieve the data from the L3 memory and send it through the UART every time the new chirp frame is being processed. Subsequently, we would be able to operate only with data connected with *some*, not time-adjacent chirp frames and such data will be significantly harder to process.
3. The proposed way of getting the data requires a computer with the CCS running to be able to retrieve the data from the L3 memory. Such setup is not suitable for being used in the standalone radar-based speedometer.

The raw data capture has therefore been deprecated and we moved our focus to another approach to the data processing.

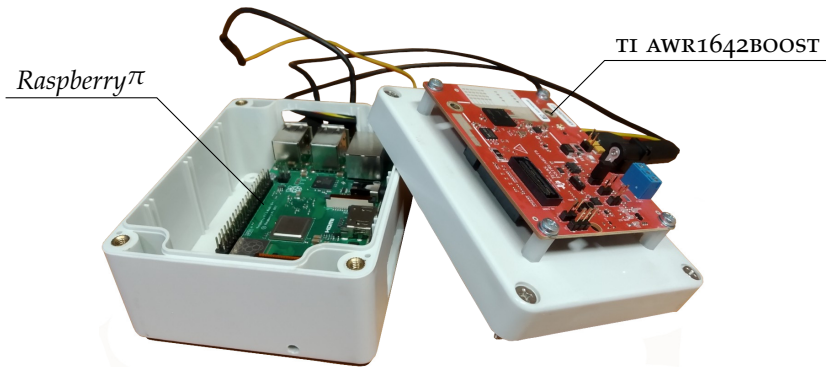
### 2.3.2 Decentralized processing

After recognizing the unapproachability of gathering the raw ADC data from the TI AWR1642BOOST without an external hardware tool, we settled on the following data processing approach:

1. Detection of objects in the field of view of the EVM directly in the EVM.
2. Sending the results to a small PC (*Raspberry $\pi$* ) via the UART.
3. Computing the precise velocity value from the data received via the UART in the small PC.



The second and the third item of the list above utilize a small PC. We have let its role to the *Raspberry $\pi$*  – a computer small in size but universal and robust in the functionality. The particular model is the 3B+ which disposes with an integrated Bluetooth adapter that we want use for delivery of the final velocity samples to an external laptop. The two “processing devices” – the TI AWR1642BOOST and the *Raspberry $\pi$*  are put near each other in the mounting box and interconnected by a data cable. The data cable connects one of four USB ports of the *Raspberry $\pi$*  and the  $\mu$ USB port of the TI AWR1642BOOST. If the chained power scheme is used (see Sec. 2.4), the two devices are interconnected also with a power cable between a USB port of the *Raspberry $\pi$*  and the EVM’s power jack. The box with the two devices is depicted in Fig. 2.15.

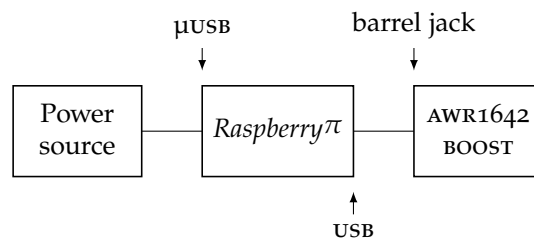


**Fig. 2.15:** The mounting box with the TI AWR1642Boost and the *Raspberry $\pi$*  (no power source included).

## 2.4 Power supply

The complete setup of a radar device contains two entities which need to be powered: The *Raspberry $\pi$*  and the TI AWR1642BOOST. The *Raspberry $\pi$*  is standardly fed by 5 V from the  $\mu$ USB<sup>vii</sup> power port (although it is possible to power it up through the pin header or to use the POE (power over ethernet) feature). The maximum current request is 2.5 A [24]. The TI AWR1642BOOST needs a 5 V power source connected to the 2.1 mm barrel jack. The current limit is 5 A [10].

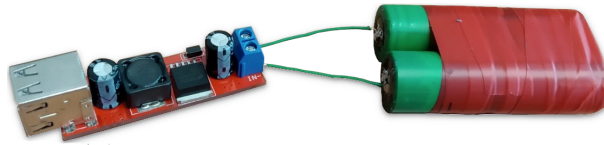
For some configurations of the EVM, powering the two in a cascade from one common power source is possible as is depicted in Fig. 2.16, where the TI AWR1642BOOST is powered from an USB port of the *Raspberry $\pi$* . Some other configurations (here belongs also the one that we eventually decided to employ) require more current than a single USB port, depleted of the *Raspberry $\pi$*  current consumption, could give. In such cases, two distinct power sources have to be used for the *Raspberry $\pi$*  and for the radar. For a practical use it is possible to connect the power pins with wires (not employing connectors) and save some space.



**Fig. 2.16:** The power chain of the radar hardware.

<sup>vii</sup>Connector USB micro-B.

As a stable voltage source, a battery of two *Sony US18650VTC6* accumulators [25] connected in series has been used together with a step-down buck converter with an USB output LM2596USB [26]. A photo of the source is in Fig. 2.17.



**Fig. 2.17:** The power source.

Such power source is of course just a solution for testing the whole device when the use of another (more robust) source is complicated. When the device was used in a car or in a laboratory, there are several other possibilities how to feed it, namely USB power from a PC, a laboratory source, or a car  $\mu$ USB charger.

## Signal and its processing

The prototype of the radar speedometer has been introduced in the previous chapter. The signals wandering in the circuit carry all the information that could be used to figure out the measured horizontal velocity. In this chapter, we will describe how the signal looks like and how it is processed both in the TI AWR1642BOOST and the *Raspberry $\pi$* . Different signal and signal processing designs will be reviewed in order to achieve the best performance of the velocity estimator.

### 3.1 The original firmware

The *mmWave demo* program (firmware) is a toolkit distributed together with the TI AWR1642BOOST EVM. The program contains drivers for the most of the EVM's functionality: It provides separate programs for the master (ARM Cortex R4F) and the signal processors, defines and monitors their cooperation, constitutes a CLI (command line interface) to allow user intervention and an online configuration, etc. Inherently, it defines the shape of the transmitted signal and the processing of the received echo. The program is well described in the *Doxygen*-based<sup>i</sup> documentation distributed with the firmware [22]. The firmware itself could be alternatively accessed as a part of the *Industrial toolbox* [27].

#### 3.1.1 Flashing the firmware to the board

The flash of the *mmWave demo* binaries could be done using the *Uniflash* tool [28] by TI. The TI AWR1642BOOST has to be connected to the PC with the *Uniflash* installed, set to the *flash programming* mode (see Tab. 2.1), and powered up. After being connected, the board manifests itself by two emulated serial ports, named differently on *Windows* and *GNU/Linux* systems:

	GNU/Linux	Windows
<b>Configuration port</b>	/dev/ttyACM0	COM3
<b>Data port</b>	/dev/ttyACM1	COM4

**Tab. 3.1:** The standard names of the two serial ports of the AWR1642Boost.

These port names need to be specified in the *configuration* tab of the application. Subsequently, the binaries have to be selected and the flashing process could be launched. The precompiled binaries are distributed with the EVM (or as a part of the *Industrial toolbox*), however if that is not the case, their source code could be compiled using the CCS IDE [29] following the steps in corresponding materials available from the TIREX database [30].

<sup>i</sup>*Doxygen* is a software tool for generating a documentation from the markup written directly to the program source code.

### 3.1.2 Configuration via the CLI

An user could control the form of transmitted radar signal and how it will be processed using the CLI (command line interface) commands sent via a terminal connected to the configuration serial port. The predefined commands are roughly defined in [31] and reviewed on the following lines. Possible arguments of the commands are delimited by *space* (ASCII 0x20); Command `cliCommand` with three arguments: 0, 5.162 and 13 has to be sent in the form

```
1 cliCommand 0 5.162 13\n
```

**Listing 3.1:** The example CLI command with three arguments.

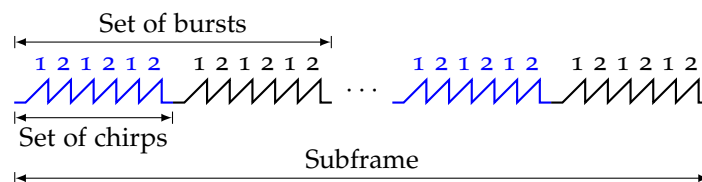
**sensorStop** command has no arguments and leads to termination of the radar sensor operation and the signal processing chain. It is a mandatory command before *any* reconfiguration is being done.

**flushCfg** is another mandatory command that has to be executed before any reconfiguration is being done – the right place for its execution is right after the `sensorStop` command. It causes the old configuration to be neglected and prepares the firmware to process the new configuration expressed in the following commands.

**dfedataOutputMode** is mandatory but has no fixed position – it could be executed anywhere between the `sensorStop` & `flushCfg` pair and the command `sensorStart` (described below). It has one argument that defines the mode of the frame configuration. The argument could get values 1 or 3. Value 1 stands for *frame-based chirps* mode, value 3 for the *advanced frame configuration* mode. In both of the modes, the *frame* (group of transmitted chirps) consists of many times repeated *subframes*. The modes differ from each other in how the subframes look like.

**In the frame-based chirps** mode the subframe consist of  $N_t$  chirps, where  $N_t$  is a number of TX antennas used. The first chirp is transmitted by TX1, the second by TX2 etc.

**In the advanced frame configuration** mode, user can define up to 4 different subframes and in each of the subframes, there could be an unique sequence of up to 512 groups (bursts) of up to 512 chirps [32]. A subframe of the advanced frame configuration mode is sketched in Fig. 3.1.



**Fig. 3.1:** An advanced chirp configuration subframe. Within the particular bursts, the chirps are transmitted alternately by TX1 and TX2 antennas (if both TX antennas are enabled).

**channelCfg** command is mandatory and we can use it to select how many TX or RX antennas are used. It requires three arguments:

1. RX antennas mask. If the antenna is to be enabled, it gets value 1; In the opposite case, it gets zero. The value of the argument is the number which digits are the enable values for RX1, RX2, RX3, and RX4, respectively, coded in decimal. If all the RX antennas are intended to be enabled, the value  $(1111_2)_{10} = 15$  has to be set.
2. TX antennas mask. The same as the RX antennas mask but for the TX antennas. Only two TX antennas are available on TI AWR1642BOOST – to enable both, one has to enter value 3, to enable only TX1, one has to enter 2.

3. The third argument is not applicable (NA). For NA arguments we will always use value 0.

`adcCfg` is a mandatory command setting properties of the ADC. In fact, there are only two possibilities for filling up its two arguments. The arguments are

1. Number of bits of the ADC. Only value 2 standing for 16bit ADC is supported in TI AWR1642BOOST.
2. Sampling scheme. The sampling schemes could be generally divided to *real* and *quadrature* (complex). The real schemes use one-mixer circuits like the one in Fig. 3.2. The spectrum

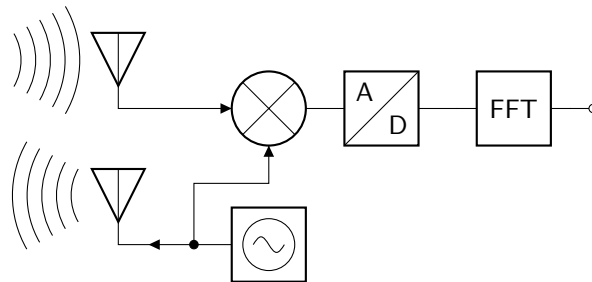


Fig. 3.2: The real mixing.

of the mixer output contains a convolution of spectra of the mixer inputs, which leads to doubling of the noise component at the output as is illustrated in Fig. 3.3. On the

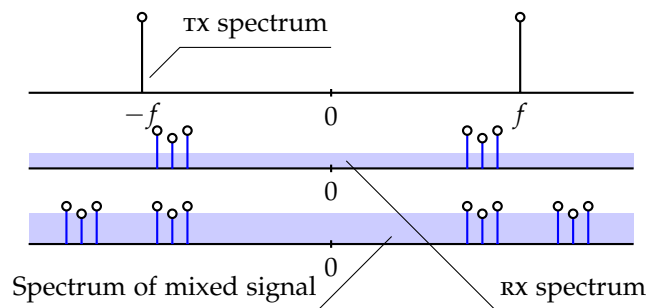


Fig. 3.3: Spectra of mixer inputs and a mixer output in the real mixing scheme.

other hand, the complex schemes use quadrature mixing. The schematic of such mixer is depicted in Fig. 3.4. Although such scheme consumes more hardware components, it provides an advantage of the lower noise present in an output of the mixer, as could be seen from Fig. 3.5. Another advantage of the complex mixing is that we can monitor the noise level simply by observing the spectrum of the mixed signal in the image band within one of the sidebands (In fact in the TI AWR1642BOOST there is always a *SSB* (single side band) spectrum used [33]). The same principle could be used to detect possible interference. In the *EVM*, only the *complex* mixing is possible. However, there are two modes of it, from those we can select one. The so-called *complex 1x* mode is the standard complex mixing as has been described a few lines above, with usable spectral samples occupying a frequency range from zero to the sampling frequency. The *complex 2x* mode uses a digital frequency shifter of the result before filtering out the unused part of spectrum. That leads to the spectrum samples occupying a frequency range from  $-f_p/2$  to  $f_p/2$ . This mode requires only half of the sampling rate of the *complex 1x* mode to fulfil Nyquist condition, however, there is no image band filtering present which causes that signal processed this way is more prone to be jammed.

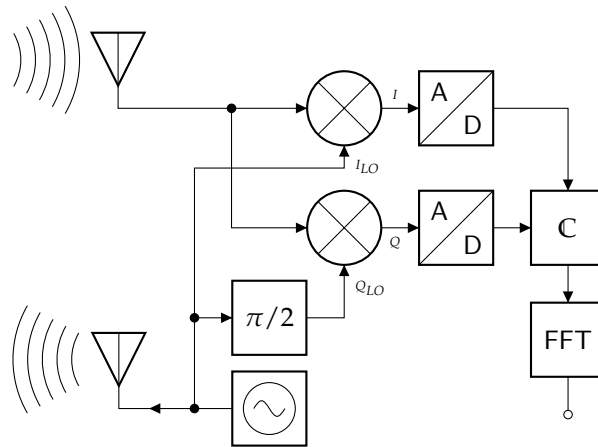


Fig. 3.4: The quadrature (complex) mixing.

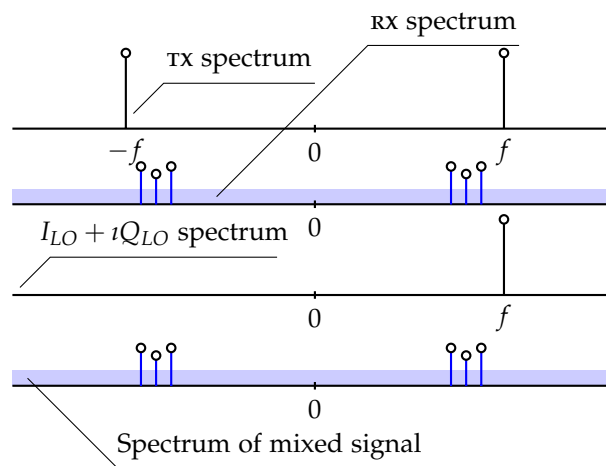


Fig. 3.5: Spectra of mixer inputs and a mixer output in the quadrature mixing scheme.

For the *complex 1x* mode, the second argument has to be set to 1. For the *complex 2x* mode, set it to 2.

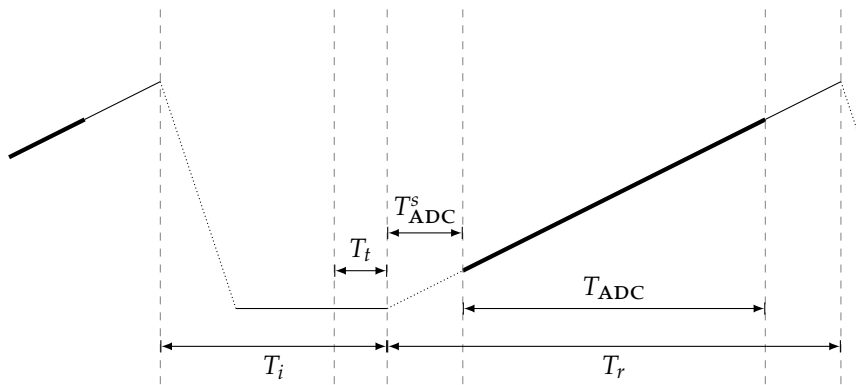
**adcBufCfg** is a mandatory command with 5 arguments allowing configuration of the ADC samples buffer. The arguments are:

1. Subframe index. This argument offers possibility to apply differend ADC buffer configurations for different subframes if the advanced frame configuration mode is used. To use the CLI command for a particular subframe, the subframe index of the subframe has to be here. If we wanted to use the same configuration of the ADC buffer for all subframes (advanced frame configuration mode is not used), the value -1 has to be entered.
2. ADC output format resolves between complex (value 0) and real (not even possible in TI AWR1642BOOST) sample format.
3. Sample swap determines positioning of the *I* and *Q* samples in one bin of the ADC buffer. Only the option "*I* first (on the LSB), *Q* second (on the MSB)," expressed by the value 0 is possible.
4. Channel interleave. For details, see [22,31]. Only value 1 is supported for the EVM.

5. The argument called *chirp threshold* stands for a number of chirps accumulated in the ADC buffer before the chirp interrupt signal is emitted, i.e. before the processing of the values in the buffer is started. The value of the argument could be from 1 (chirp interrupt emitted after every single chirp) to 8 (at most 8 chirps in the buffer waiting to be flushed). Setting more chirps could help the processing to be more efficient. There is also a special value of this argument—zero—that means “Set the maximum possible chirp threshold.”

**profileCfg** is a mandatory command with 14 arguments setting up chirp signal and timing parameters for one *subframe*. A subframe is one block of a *frame*. In one frame there could be the same subframe repeated many times (the standard (*legacy*) mode) or up to four different subframes in a row, repeating as an ordered sequence in a frame. For more details, see the description of the `dfcDataOutputMode` CLI command above, or [31].

Before the arguments are listed, the nomenclature should be explained. In the time – frequency diagram of a pair of adjacent chirps (Fig. 3.6) all the variables used later in the description of arguments are expressed.



**Fig. 3.6:** Two adjacent chirps of one subframe.  $T_i$  is the *idle time*,  $T_r$  the *ramp time*,  $T_{ADC}$  the *ADC sampling time*,  $T_{ADC}^s$  the *ADC start time*, and  $T_t$  the *transmission start time*.

The arguments of the CLI command are:

1. Profile identifier. For the standard frame configuration (all subframes equal), there has to be only one `profileCfg` command within a configuration file and the argument could get *any* value. For the advanced frame configuration mode, there has to be one `profileCfg` command for every subframe and the argument could get an arbitrary value, but different for each subframe. All the following arguments are proper to *one subframe*.
2. Start frequency of the chirp in GHz. Could get any value from 76 to 81.
3. Idle time in  $\mu\text{s}$ . Could get any value from 0 to 524287.
4. ADC start time in  $\mu\text{s}$ . Any value from 0 to 4095 is allowed.
5. Ramp end time in  $\mu\text{s}$ . Values from 0 to 500000 are allowed.
6. (NA) TX output power backoff.
7. (NA) TX phase shift.
8. Frequency slope of the chirp in  $\text{MHz } \mu\text{s}^{-1}$ . Could get values from -2072 to 2072.
9. Transmission start time in  $\mu\text{s}$ . That is time between start of a transmitter and the start of the ramp. Values from -4096 to 4095 are permissible.
10. Number of ADC samples taken during the ADC sampling time  $T_{ADC}$ . Could get value of any multiple of 4 from 64 to  $N_{s,\text{max}}$ , where  $N_{s,\text{max}} = \frac{\text{size of sample memory}}{(\text{size of one sample})(\text{number of rx antennas})} = \frac{16384}{4N_r}$ . For 4 RX antennas, the maximum allowed value is 1024.

11. ADC sampling frequency ( $f_p$ ) in kHz. The valid range is from 2000 to 37500.
12. The first HP filter cutoff frequency. Between the mixer and the sampler (ADC), there is a cascade of two high pass filters to suppress the low frequency signals. There are four options for the cutoff frequency for each of them. For the first filter, setting the argument to 0 means 175 kHz, 1 means 235 kHz, 2 means 350 kHz, and 3 is 700 kHz.
13. The second HP filter cutoff frequency. The same description as 12., but different mapping of argument values to cutoff frequencies: 0 means 350 kHz, 1 means 70 kHz, 2 means 1.4 MHz, and 3 is 2.8 MHz.
14. Gain of a vga (variable gain amplifier) placed after RX antennas expressed in dB. Allowed values range from 24 to 52.

**chirpCfg** is another mandatory CLI command. It takes 8 arguments and serves for setting the basic parameters of chirps in particular subframe. For each subframe, there could be many **chirpCfg** commands. Each one of them sets the parameters for chirps from *start index* to *end index*. Therefore, each subframe may contain many bursts of chirps varying in the chirp parameters. The arguments are:

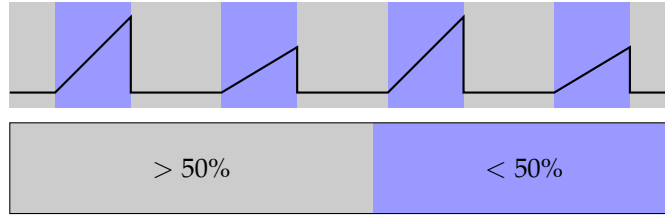
1. Chirp start index. Could get any integer value from 0 to 511. Its value has to be lower or equal as the value of the next argument, chirp end index.
2. Chirp end index. Allowed are integer values from the value of chirp start index (inclusively) to 511.
3. Profile identifier. The value has to be the same as the value of the profile identifier in the **profileCfg** command corresponding to the adjusted subframe.
- 4-7. NA.
8. TX antenna enable mask. The chirp could be transmitted by TX1 only, TX2 only, or we do not care and the two antennas could be used for transmission alternately. The binary mask has two digits: the first one is for the TX1 and the second for TX2. Digit 1 means "allowed," digit 0 "not allowed". The value of the argument is a decimal representation of the binary mask. For example, if only antenna TX1 was allowed, the argument would get the value  $(10_2)_{10} = 2$ .

**frameCfg** is a command that is mandatory only for the standard (legacy) frame configuration mode. In that mode each frame contains only one type of subframe. The subframe is repeated until the number of repetitions meet the value proper to the frame. The subframe is made of at maximum 512 chirps those do not have to be all the same. Number of subframe repetitions within a frame could be at most 255. The command's 7 arguments are:

- 1.-2. Chirp start index and chirp end index, respectively. The allowed values and the logic behind is exactly the same as with the **chirpCfg** command.
3. Number of repetitions of the subframe within a frame. The allowed values are powers of 2 between 16 to 255.
4. Number of frames overall. After transmission of that many frames, the EVM stop transmitting at all. The value could be set to any integer from 1 to 65535, inclusively, or to zero to indicate *infinite* transmission.
5. Frame duration in ms. The allowed values are from 1 to 1000, however, it has to be ensured that the duty cycle of the frame is less than 50%, i.e. if the overall duration of chirps in one subframe set by the **chirpCfg** command is  $t_c$  and the number of repetitions of a subframe within a frame is set to  $n_r$ , the value of this argument should be greater than  $1000(t_c n_r / 2)$ . For an explanation of the *duty cycle* of a frame, see Fig. 3.7
6. NA. Has to be set to 1.



7. Frame trigger delay in ms. According to [31], the delay could avoid interference if more than one sensors of the same kind as TI AWR1642BOOST are employed near each other. Allowed values range is not listed but we expect the typical values to be from 0 to *few tens of microseconds*. In all of the example configuration files this argument gets zero.



**Fig. 3.7:** A frame. A duty cycle is a ratio between the overall duration of chirps and the overall duration of the idle time.

**LowPower** is a mandatory CLI command with two arguments of the fixed form (in case of TI AWR1642BOOST board). Since the first argument has no meaning and should be set to 0, and only the *low power ADC mode* is supported, the two arguments are always 0 and 1, respectively.

**guiMonitor** is mandatory and sets which data are sent to a user via the UART. All arguments but the first one are binary with the positive logic (i.e. 1 means to transmit, 0 means not to).

1. Subframe index. If the advanced frame configuration was used, this value has to match the subframe index value of the adjusted subframe (see the *profile identifier* argument of the `profileCfg` command). If legacy frames are used, this value has to be set to -1. If the same `guiMonitor` configuration is intended to be used for all the subframes of the advanced frame configuration, set this value to -1 as well.
2. Detected objects. For each of the detected objects in the radar's field of view the detected objects structure contains its range index, Doppler index, peak value, and spatial ( $x, y, z$ ) coordinates. The range and Doppler indices could be transformed to range and velocity by multiplication by constants  $C_r$  and  $C_d$ , respectively. The constants could be computed as [22]

$$C_r = \frac{c_0 f_p}{2SN_{\text{FFT}}} \quad (3.1)$$

$$C_d = \frac{c_0}{2f_1(T_i + T_r)N_c}$$

where  $c_0$  is velocity of EM wave's propagation,  $f_p$  is sampling frequency,  $S$  is a frequency slope of the chirp,  $N_{\text{FFT}}$  is number of range bins (the 10<sup>th</sup> argument of the `profileCfg` command),  $f_1$  is the starting frequency of the chirp in the subframe (the second argument of the `profileCfg` command),  $T_i$  and  $T_r$  are idle time and ramp time, respectively (the third and fifth argument of the `profileCfg` command), and  $N_c$  is number of chirps per frame.  $N_c$  could be computed as number of subframes in frame (the third argument of the `frameCfg` command) multiplied by overall number of chirps in the subframe. The number of chirps in a subframe is given as

$$\sum_{i \in \text{chirpCfg}} (\text{stop index})_i - (\text{start index})_i. \quad (3.2)$$

In (3.2) the iteration is proceeded along the `chirpCfg` commands proper to the examined subframe.

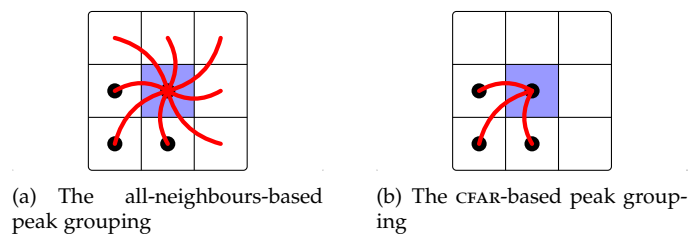
The *peak value* is 2D-FFT peak value of the cell at the range index and the doppler index. The value is unitless and the bigger it is, the stronger the significance of the object at the given range index and velocity index is. Only objects which passed the *peak grouping*

procedure are transmitted within the structure. There are two possible peak grouping schemes according to [22]: For both of them, the following is proceeded:

1. The cell under test (CUT) is put to the center bin of the  $3 \times 3$  grid.
2. The rest of the grid is filled by the cells neighbouring to the CUT (see Fig. 3.8).

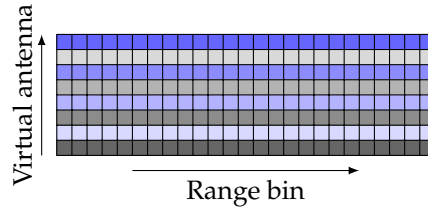
Then the CUT does pass the peak grouping test (is said to be peak), if its 2D-FFT value is higher than any of the *reference cells*. The two implementations differ in the reference cells selection. For the all-neighbours-based implementation (Fig. 3.8(a)), the reference cells are all cells in the close vicinity of the CUT. For the second—CFAR-based—implementation (Fig. 3.8(b)) the reference cells are only those cells those passed the CFAR algorithm. CFAR (constant false alarm rate) will be explained later in the description of the `cfarCfg` CLI command. The spatial coordinates' values are in meters.

3. Range profile. These are the FFT values of the row of range – Doppler matrix with zero Doppler index. The range profile vector has high values (peaks) at the positions corresponding to ranges of detected objects. Two objects at the same range cannot be resolved in the range profile. Since there are multiple virtual antennas present, this value is a base-two logarithm of the elementwise product of the values for particular virtual antennas.
4. Noise profile. It is assumed that there would not be any objects in the field of view, which velocity falls into the highest Doppler bin. Consequently, we could expect the row of the range – Doppler matrix with the highest Doppler index to contain only the FFT values of noise. The handling of the multiple virtual antennas case is the same as with the range profile.
5. Range – azimuth heatmap. This is the content of the zero-Doppler range – Doppler matrix rows for all the virtual antennas, as is depicted in Fig. 3.9. To extract the azimuth information from hence it is necessary to compute the DFT along the virtual antenna axis.
6. Range – Doppler heatmap. The range – Doppler heatmap is the computationally most demanding structure of the original firmware's portfolio. It contains the full 2D-FFT image of the scene. It is divided to range bins in one dimension, and to Doppler bins in the second dimension. If multiple virtual antennas are allowed, the values of the overall (sent) range – Doppler heatmap are the base-two logarithms of the elementwise product of the range – Doppler heatmaps of particular antennas.
7. Statistics of the processing. The structure contains some information about timing of the processing, or information about the load of the CPU.



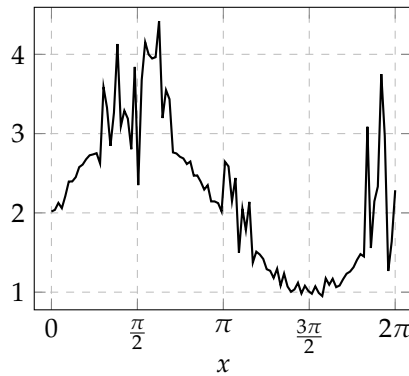
**Fig. 3.8:** The two possible schemes of peak grouping in the original radar firmware. The blue cell is the CUT, the cells with the black dots passed the CFAR detection. The red lines mark comparison between the FFT values of cells.

`cfarCfg` is mandatory and determines parameters of CFAR (constant false alarm rate) processor for given subframe. CFAR is a technique arranging for constant rate of false feature detections in signal [34]. The idea behind it is to monitor the noise power at a vicinity of each signal sample, and adjust the detection threshold according to it. Assume the signal from Fig. 3.10. It is obvi-



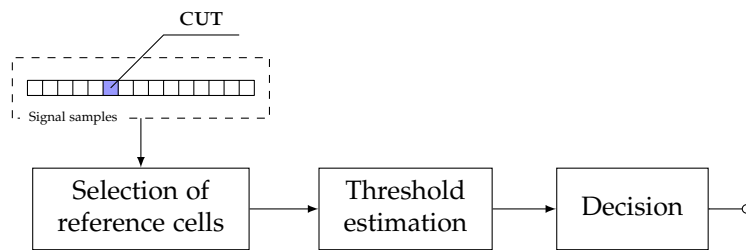
**Fig. 3.9:** The range – azimuth heatmap structure. The cells contain complex ADC samples.

ous that the detection of the peaks with no additional signal processing<sup>ii</sup> cannot be satisfactorily done with one overall threshold. The CFAR processing is essentially an *adaptive processing* technique with a specific goal. The block diagram of general CFAR detection is depicted in Fig. 3.11.



**Fig. 3.10:** Peaks superimposed on the sine signal with AWGN.

The input signal is assessed sample by sample and the examined sample is denoted as CUT<sup>iii</sup>. In the *Selection of reference cells* block, an algorithm has to be implemented that determines which of the cells other than the CUT are employed in a calculation of the threshold. The block of *Threshold estimation* estimates the most appropriate threshold level for the CUT out of the cells selected by the preceding block. The terminating block then just decides whether the feature *is* or *is not* present in the CUT according to the estimated threshold level.



**Fig. 3.11:** The block diagram of general CFAR detection.

According to [22], there are 3 particular CFAR schemes implemented in the original firmware:

**CA-CFAR** (Cell-averaging CFAR) is the simplest of them and will be explained with reference to Fig. 3.12. In the picture there is a signal vector which peaks we would like to find. For each

<sup>ii</sup>Knowing that the peaks are superimposed on the sine signal with the known frequency and amplitude, that will be not a hard task to detect the peaks without employing any CFAR technique.

<sup>iii</sup>The name *cell* is more appropriate than the name *sample* here since the input signal could be generally  $n$ -dimensional.

CUT (sample), the active cells (the cells contributing to the result for the CUT) are defined by the maximum and minimum distance from the CUT. The active cells are filled by light blue in Fig. 3.12. From them an average is calculated and the value of the CUT is compared with the average. If the value of the CUT was lower than the average, then the null hypothesis (There is not a peak in the CUT) is confirmed; Otherwise, we declare that there is a peak present in the CUT. The cells nearer to the CUT than the minimum allowed distance are called *guard interval*. The cells near the boundary of the sample vector could be handled many ways, however, the firmware enables only these two [22]:

**Cyclic mode**, with which the sample vector is assumed to be cyclic (so the first and the last cell are connected). The number of active cells and the number of cells in the guard interval are constant for each CUT.

**Non-cyclic mode**, that simply do not use the cells those are not available and computes the average from the available ones.

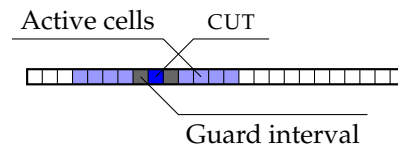


Fig. 3.12: CA-CFAR.

**CAGO-CFAR, CASO-CFAR** (Cell-averaging greatest/smallest of CFAR). These are the techniques suppressing two weak spots of the traditional CA-CFAR, both occurring in situations where the CUT lies on the border between two different noise/clutter regions [35]:

**Considering a clutter to be a peak of the useful signal.** The CAGO-CFAR could help here.

**Not detecting a peak because of too strict threshold.** The CASO-CFAR could help.

With them both the guard interval works the same way as in the traditional CA-CFAR. However, the mean is computed separately from the cells on the left side of the CUT and from the cells on the right side of the CUT. The threshold is set as the greatest of the two in the CAGO-CFAR, and as the smallest of them in the CASO-CFAR.

In the detection matrix (see 1.3.4) the CFAR detection could be done in two directions – for each of the rows (range dimension) or for each column (Doppler dimension). When the CFAR is already performed in columns/rows of the matrix, the CFAR in the second dimension could be performed in only those rows/columns which contain the detected peaks. The CLI command itself has 8 arguments:

1. Subframe index. The same rules hold as for the *subframe index* arguments in the previous CLI commands.
2. Processing direction. Zero indicates *range* direction, one the Doppler direction. There have to be exactly 2 *cfarCfg* commands for each subframe: One for each of the processing directions (range/Doppler).
3. CFAR mode. Zero stands for the CA-CFAR, 1 for the CAGO-CFAR, and 2 for the CASO-CFAR.
4. Number of the active cells in one side of the CUT.
5. Number of the guard cells in one side of the CUT.
6. Noise sum divisor. When the average value of the noise cells (active cells) is calculated, the values are first summed and then divided by the noise sum divisor, i.e. the number of active cells which were summed. In the CA-CFAR mode, the value is double the fourth argument, in the CASO- and CAGO-CFAR it is exactly the value of the fourth argument. Nevertheless, the *cfarCfg* command needs the value of the divisor expressed in *number of*

positions to shift the binary representation of the sum by. The value could be computed from the divisor  $d$  as  $\log_2(d)$ . That also implies the need for selecting the overall number of active cells as a power of 2.

7. NA.
8. Detection margin over the cell average. This is a value  $M$  by which the average of the active cells are multiplied to obtain the threshold. The `CUT` is then declared to be a peak if it outgrows the threshold. The value of this argument is not very straightforward and could be computed as [31]

$$M_{\text{CLI}} = \frac{256 \cdot 20 \log(M) N_r N_t}{6} \quad (3.3)$$

where  $M_{\text{CLI}}$  is the number to be entered as the argument, and  $N_r$  and  $N_t$  are number of rx and tx antennas, respectively.

**peakGrouping** is another command mandatory for each subframe. It configures the *peak grouping* scheme which is a tool for reducing peak clusters to standalone peaks. Depending on the range and Doppler resolution and the overall measurement goal, the peak grouping could be enabled or disabled. There are two algorithms for the peak grouping implemented in the original firmware – the *all neighbours* peak grouping and the *CFAR-only* peak grouping. For their descriptions see the description of the second argument of the `guiMonitor` command or [22].

The CLI command requires these 6 arguments:

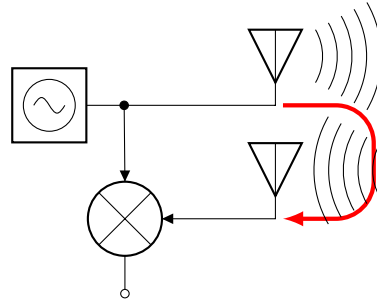
1. Subframe index. See the *subframe index* argument description of the `guiMonitor` command.
2. Peak grouping scheme selection. Number 1 means the all neighbours scheme, number 2 is for the *CFAR-only* scheme.
- 3., 4. Enable flag for peak grouping in the range and the Doppler directions, respectively. 1 means to enable it, 0 means to disable it.
- 5., 6. Maximum and minimum range index to consider, respectively. The cells with index lower than the minimum index or higher than the maximum index are not considered at all – even if there were peaks detected on those positions, they would be neglected.

**multiObjBeamForming** configures a simple algorithm allowing the processing to distinguish between two objects at the same range and Doppler bin. The only way how to resolve them is to employ the AOA processing (see 1.3.6). The algorithm implemented in the demo firmware observes the result of the DFT along the virtual antennas dimension. If the highest peak value of that was  $F_{\text{max}}$  and the second highest value is higher than  $C_{\text{MO}} F_{\text{max}}$ , then the object in the corresponding range/Doppler position is put to the resulting list of objects *twice*. The  $C_{\text{MO}}$  is a constant set by one of the arguments of this command. This algorithm allows only for detection of less or equal than 2 objects of the same range/Doppler bin. The arguments are:

1. Subframe index – the same subframe index as with many of the previously described commands.
2. Enable swith. Given 0, the feature is disabled and only one object is detected for each of the range/Doppler bins. Given 1, the detection of the second object is allowed.
3.  $C_{\text{MO}} \in \langle 0; 1 \rangle$ .

**clutterRemoval** is a mandatory command for configuration of the very simple static-clutter-removal algorithm. If such processing was enabled by the second argument of this command (binary, positive logic), then for each vector entering the FFT (in both range or Doppler dimension), its mean value is subtracted. The first argument is the *subframe index*.

**calibDcRangeSig** is mandatory and serves for calibration of the interference in the range dimension caused by coupling between the TX and RX antennas. When the signal from the TX antenna approaches the RX antenna untimely (before it is reflected from a reflector), the mixer outputs an unwanted DC signal (see Fig. 3.13). The signal of near-to-DC frequency is also produced by objects in close vicinity of the radar. With no objects occupying certain range bins present it is possible to calibrate the DC bias by subtracting the mean of signal in the those range bins from all the other range bins. That is exactly what the algorithm implemented in the firmware does.



**Fig. 3.13:** The coupling between TX and RX antennas.

It requests these arguments:

1. Subframe index. See the description of the *subframe index* argument with the `guiMONITOR` command.
2. Enable switch. 0 to disable the calibration, 1 to enable it.
- 3, 4. The index of the highest of the low range bins ( $I_1$ ), and the negative index of the lowest of the high range bins ( $I_2$ ) to serve as a source of the background, respectively. It is assumed that in the first ( $I_1 - 1$ ) (zero-based indexing) and in the last ( $-I_2$ ) range bins there are no objects present. From these boundary bins, the background is calculated as their mean value.
5. Number of chirps inputting the calibration process. The calibration does not have to be provided continuously. The background could be estimated from the first chirps at the beginning of measurement, and then also the boundary bins employed in its calculation become usable. According to [31] this argument has to get a power of 2.

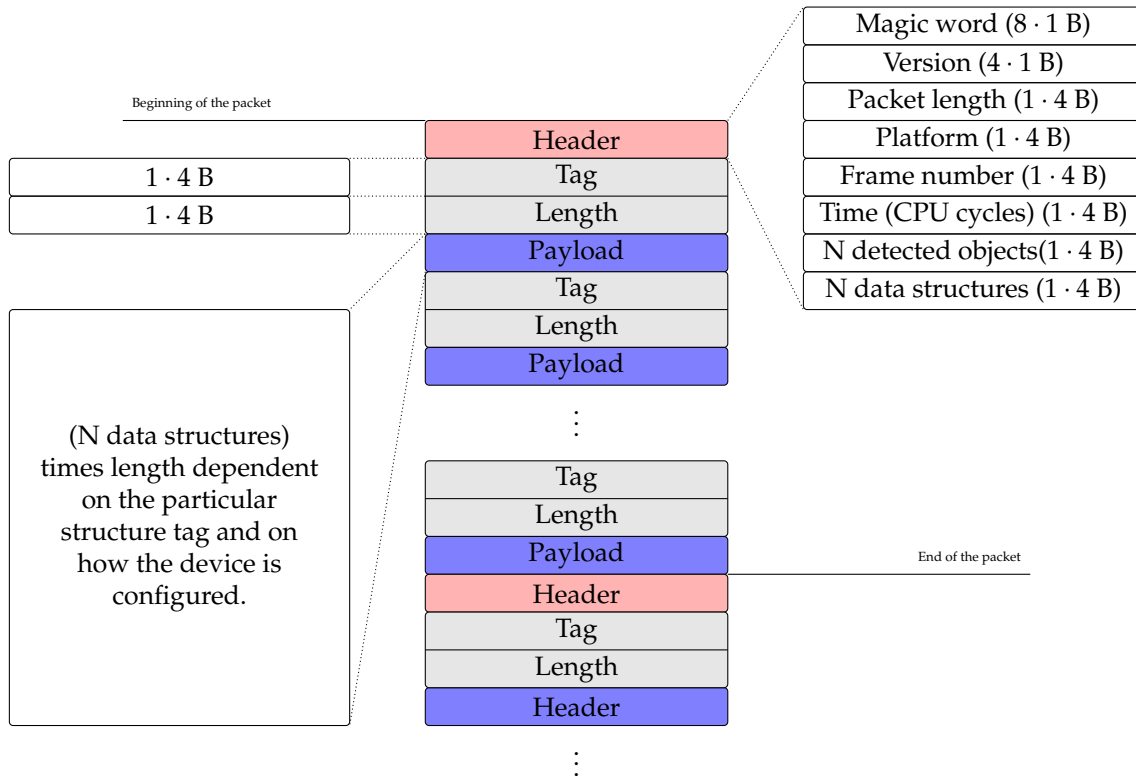
**Other mandatory CLI commands.** These contain commands `extendedMaxVelocity`, `compRangeBiasAndRxChanPhase`, and `measureRangeBiasAndRxChanPhase`. The first of them is there to suppress an ambiguity in Doppler estimation and therefore to allow for higher maximum unambiguous velocity. The second and the third are used for calibration of the channel phase. These commands were not addressed in our design and we used directly their forms from the example configuration file distributed with the MMWAVE SDK [31]. For their descriptions, see [22,31].

**sensorStart** is the last mandatory command. It closes the configuration and turns on the transmission, reception, and processing based on the CLI commands between it and the `sensorStop` command.

### 3.1.3 Received data format

The processed data structures are sent via UART as a binary stream with packetized information. The content of the stream depends on arguments of the `guiMonitor` command, however, its structure is fixed and displayed in Fig. 3.14.

The description of the particular fields could be found in the [22]. Since the original documentation is split into documentation of several data structures and one can spend a lot of time putting the parts together, we find it advantageous to summarize the structure by few tables in appendix A.



**Fig. 3.14:** The structure of the UART data stream and the packet.

Together with Fig.3.14 the tables give a complete set of descriptions needed to read an UART data stream based on the demo firmware. An implementation of an UART data parser based on these structure descriptions is commented in Sec.4.1.1.

### 3.2 Algorithms for velocity evaluation

Not all the quantities addressed in Chapter 1 are influenced by velocity of targets. In fact, there are only two of the messages which contain velocity information: the *detected objects* list, and the *range – Doppler heatmap*. In this section, the algorithms for its extraction out of these UART structures are described.

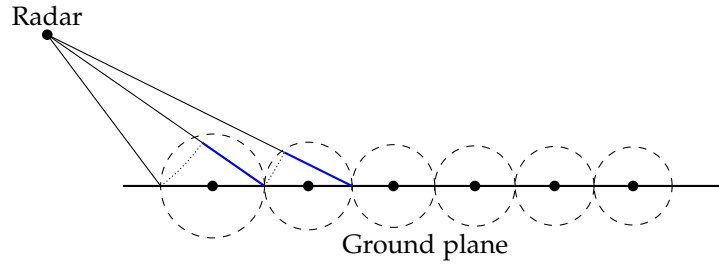
#### 3.2.1 Mean of Doppler bins argmax

This algorithm uses *range – Doppler heatmap* data.

When the radar is travelling horizontally over the perfectly flat ground plane, Doppler shift influences spectrum of *each* of the range bins. The ground plane behaves as a continuous array of elementary reflectors. From the perspective of the radar, elementary reflections sum up and the ground plane is equivalent to an array of *joint reflectors* corresponding exactly to range bins given by the chirp train configuration. In the case of intrinsic radar antenna (spherical radar wave), the joint reflectors are not uniformly spaced; The geometry is explained by Fig.3.15. In that case, the more distant a joint reflector, the lower is power of reflected signal, and the smaller is an area related to the joint reflector. Both of these facts lead to decrease in spectral peak magnitude with increasing range bin. As a consequence, also PAPR<sup>iv</sup> decreases because the noise floor does not change over the range bins.

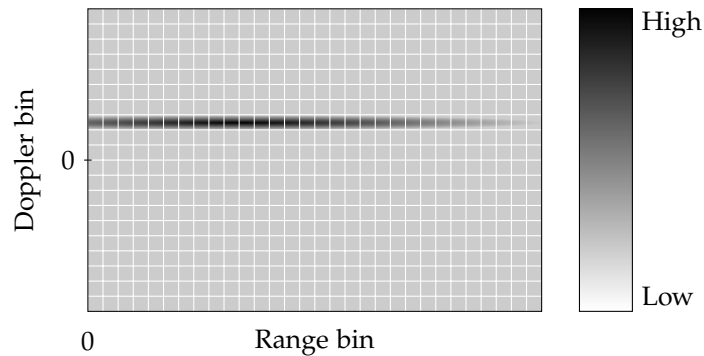
<sup>iv</sup>See Sec. 3.3.6.





**Fig. 3.15:** Geometry of a radar with an intrinsic antenna moving horizontally above ground. The joint reflectors with centers marked by black points correspond to range bins. Note that difference between distance from the radar to the closest point and to the furthest point of each joint reflector area marked by blue line is constant.

In case of directional antenna (that is the case of  $\pi$  AWR1642BOOST) the decrease of PAPR with range is even more observable. The presence of the same Doppler shift over all range bins is, however, present and the range – Doppler heatmap is expected to look like the one in Fig. 3.16.



**Fig. 3.16:** Expected range – Doppler heatmap for a case of radar moving horizontally over the ground with positive velocity.

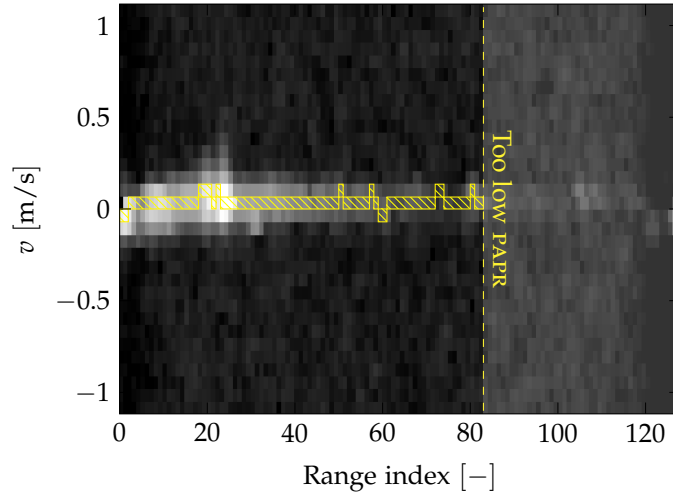
A straightforward approach is to evaluate an argument of Doppler maximum for each range bin and calculate mean of this Doppler argmax vector. The approach results in a fractional Doppler bin reflecting the horizontal velocity of the radar. A range – Doppler heatmap measured by  $\pi$  AWR1642BOOST with Doppler maxima highlighted is in Fig. 3.17. Since in the higher range bins a peak is hidden in noise, it is important to restrict the considered range of range bins to the lower range parts. The highest range bin to be considered could be given by thresholding a moving mean of peak to average power ratio (see Sec. 3.3.6), or—more practically in case when only one fixed chirp train configuration is used—by setting it to a fixed value. A *Python* example of the former is provided in Lst. C.2.

There are basically two problems to solve before using this algorithm. The first of them is to choose an appropriate range of range indices from which the mean of Doppler bin argmax ( $\mu_D$ ) is computed. Since the appropriate range depends on the chirp configuration employed, we selected the right range *after* finding out the right configuration in Sec. 3.3. The second is to find the dependence of  $\mu_D$  on horizontal velocity of the radar which is addressed in Sec. 3.4.

### 3.2.2 Mean of spectral centroid of Doppler vectors

This algorithm uses *range – Doppler heatmap* data.





**Fig. 3.17:** Real range – Doppler heatmap with Doppler maxima highlighted. In higher range bins a peak is hidden in noise.

An approach similar to the previous uses *spectral centroid* to find a peak in the Doppler vectors (columns of a range – Doppler matrix). Spectral centroid is simply centroid, with spectrum samples as an input vector. In terms of Doppler vectors  $d$  directly in frequency domain (they arrive in that form from the EVM), the spectral centroid  $\Lambda$  (unit: index) is defined as

$$\Lambda = \frac{\sum_i d_i i}{\sum_i d_i}, \quad (3.4)$$

where  $d_i$  is the  $i^{\text{th}}$  element of the doppler vector  $d$ . The iterating index in summation  $\sum_i$  iterates over all indices of the input vector  $(0, 1, \dots, \text{length of } d - 1)$ .

The use of centroid instead of direct maximum could help to

- Lower rate of erroneous maxima selections when peaks are not perfectly “sharp” (occupy more than one Doppler bin each).
- Suppress impact of strong reflectors at unexpected ranges, e.g. bumps on a conventionally flat ground plane.

Several experiments were performed to see how the centroid-based statistic for velocity evaluation acquired from a RDH differs from that of the Doppler-bins-argmax-based one. The experiments revealed the centroid-based statistic mined from a RDH without any modification is generally too little sensitive to horizontal velocity changes. It is needed to transform the Doppler vectors to mark off the high values of the Doppler vectors and their low values. Such transformation brings extra computational requirements to the processing chain. On the other hand, the benefits of  $\Lambda$ -based evaluation are not that remarkable because the Doppler bins argmax evaluation computes *mean* over all the range bins that itself makes the results less sensitive to ground plane bumps or erroneous maxima selections.

### 3.2.3 Mean Doppler of detected objects

This algorithm uses *detected objects* data.

More than algorithm, this is only another source of information about Doppler content in the field of view of the radar. According to Sec. 3.1.2, the *detected objects* structure in the UART packet contains not only spatial coordinates of the detected objects, but also their range bin and Doppler bin. The detected objects are the objects (range – Doppler matrix cells) which pass the CFAR and *peak*

grouping scheme defined by configuration. The fact yields that there is less information in *detected objects* when compared to *range – Doppler matrix*, however, it disposes with an appreciable advantage of a significantly lower size needed to be sent over UART. Referring to Sec. 3.2.1, the setup with a radar targeting a ground with some depression angle  $\alpha$  leads to a RDH with a “ridge” occupying a small number of doppler bins and many range bins. The matrix enters CFAR processing that—together with the peak grouping scheme following right after it—selects *some* of the cells and states that there are objects present on their respective positions. Because we cannot generally say which cells of the ridge are the important ones and which are negligible, the CFAR threshold scale in range direction has to be set low enough. As a consequence, most of the cells of the RDH will pass the range-CFAR. The second dimension CFAR then has to let the ridge untouched, and, if possible, discard all the other values. That will, however, require one of adaptive CFAR techniques. Not only that an adaptive CFAR scheme is not present in the original firmware but it will require that much processing and computation, that the advantage of faster UART transmission caused by smaller size of *detected objects* packet than that of *range – Doppler heatmap*, will be lost. We do not use the *detected objects* message in the final implementation of velocity evaluation.

### 3.3 Chirp train design

Performance of the algorithms from the previous section (3.2) depends on the design of the chirp train used. Although the general shape of a pulse (an *upchirp*<sup>v</sup>) cannot be modified using the original firmware, the firmware’s CLI provides good options for adjusting the timing of the chirps and spaces between them, and combining various chirps to the final pulse train. Here, the chirp parameters are reviewed in terms of its impact to the range – Doppler matrix as the structure on that the Doppler bins argmax velocity evaluation algorithm is based. A theoretical insight to their impact will be given as well as real examples from measurements. Refer to the Chapter 1 for some of the explanations of conclusions given in the following subsections.

#### 3.3.1 Start frequency of the chirp

Assuming a perfect HW, range estimation should not be influenced by the starting frequency in any way. A distance estimate depends only on frequency of the downconverted signal. Moreover, the LP filters in the receiver should suppress signal of any frequency falling to the starting frequency allowed range.

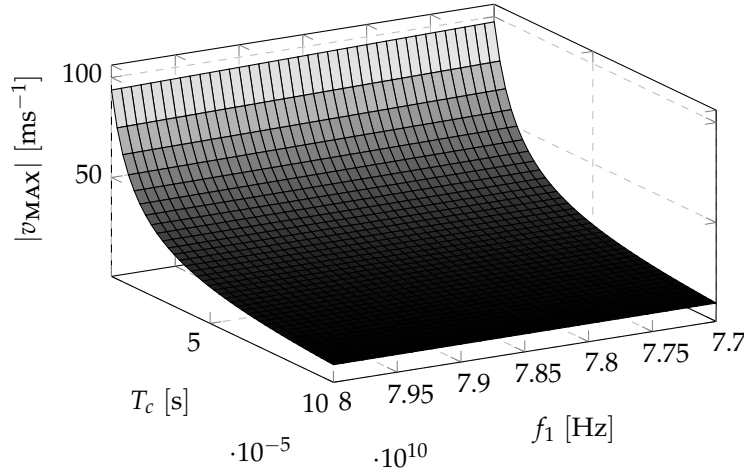
The velocity estimate depends directly on the starting frequency in both the rigorous (1.20) and the simplified (1.22) approach. Nevertheless, the velocity estimate is based on an observation of the phase difference between adjacent chirps and the change in starting frequency influences only the maximum unambiguous velocity. Referring to (1.23) the maximum unambiguous velocity  $|v_{\text{MAX}}|$  does not change significantly in the direction of  $f_1$  axis, which could be seen in Fig. 3.18.

Neither the wave propagation is influenced by the starting frequency in a significant manner: The wavelength difference between the 76 GHz and the 81 GHz (border frequencies of the allowed range for TI AWR1642BOOST) waves is 0.243 mm with the respective wavelengths being 3.947 mm and 3.704 mm.

Experiments confirmed the assumptions: No significant difference in performance of a range – Doppler heatmap has been observed between cases with various starting frequencies between 77 GHz and 79 GHz. All the other parameters remained constant throughout the cases. Higher starting frequencies were not considered since very low frequency slope (causing too high maximum discernible range) would have been needed then in order to fit the whole chirp within the frequency range of the TI AWR1642BOOST.

---

<sup>v</sup>The *upchirp* is a common name for a chirp with a positive frequency slope; The opposite is called *downchirp*.

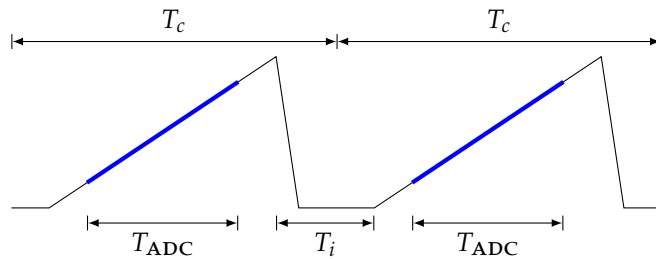


**Fig. 3.18:** Dependence of maximum unambiguous velocity on starting frequency of a chirp and the chirp duration.

### 3.3.2 Idle time

In all equations of the *Velocity estimation* section (1.3.3), The interchirp period  $T_c$  has been assumed such that the particular chirps follow immediately after each other, without any interchirp gap. That is not the real case though. In case of a single transmitter, it is impossible to achieve a frequency shift of the chirp bandwidth in zero time, as well as it is not desirable even in the multiple rx case<sup>vi</sup>.

The *idle time* parameter therefore has an impact on maximum unambiguous velocity as well as on velocity resolution. As is sketched in Fig. 3.19, the interchirp period  $T_c$  covers the idle time  $T_i$ , the ADC sampling time  $T_{ADC}$ , and the complement of  $T_{ADC}$  to the full ramp time.



**Fig. 3.19:** On impact of idle time on estimated velocity.

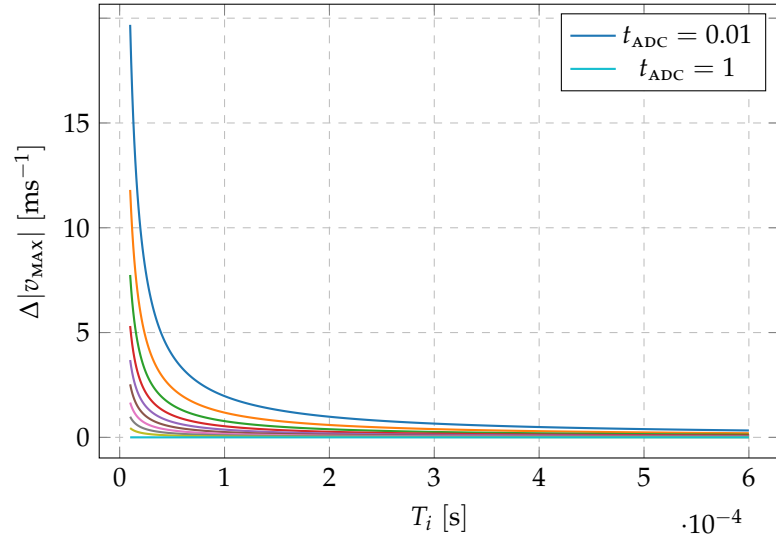
According to the description of the `frameCfg` CLI command (see section 3.1.2) at least half of the total transmission time has to be idle. Assume that the sampling is carried out during the whole ramp so (referring to Fig. 3.19)  $T_c = T_{ADC} + T_i$ . If we express the ADC sampling time as a fraction of the idle time  $t_{ADC} \in (0; 1)$ , we obtain

$$T_c = T_i(1 + t_{ADC}). \quad (3.5)$$

Having  $t_{ADC}$  fixed, we can observe the shift in  $|v_{MAX}|$  compared with the case of 50% duty cycle, i.e.  $t_{ADC} = 1$ .

Generally, prolongation of idle time *relatively* to ramp time with overall chirp time fixed leads to steeper frequency slope and therefore lower maximum discernible range (1.15). Its *absolute* prolongation causes maximum unambiguous velocity be *lower* and therefore velocity resolution being better. Obviously, a side effect of that is slowdown of the whole processing. To conclude that, for velocity

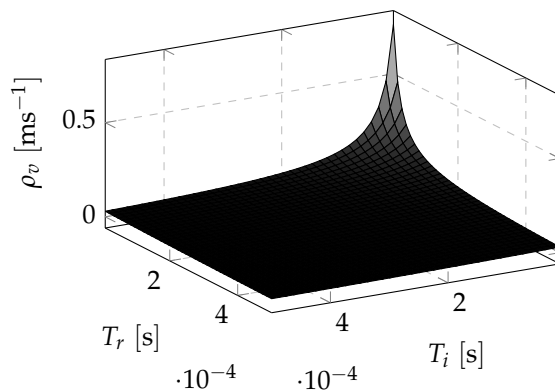
<sup>vi</sup>Even if the zero time delay between adjacent chirps was achieved using a time overlap of respective transmissions of TX1, TX2, etc. it will not be achieved in the RX signal.



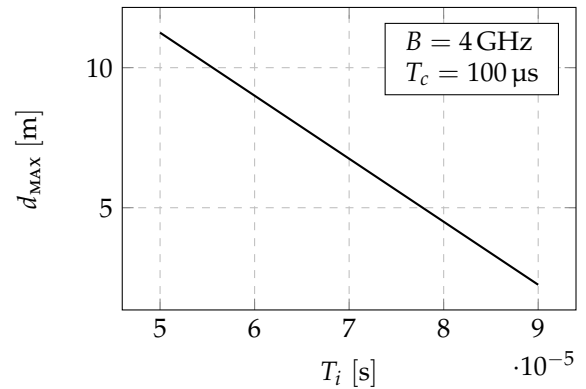
**Fig. 3.20:** The dependence of difference between maximum unambiguous range between  $t_{\text{ADC}} = 1$  and lower  $t_{\text{ADC}}$  on  $T_i$  for various lower  $T_i$  values.

estimation it could be advantageous to prolong idle time at the expense of shorter ramp. With relatively shorter ramp (assuming  $T_{\text{ADC}} = T_r$ ) we can accomplish the MUR fitting exactly in the maximum useful range resulting from an area of reflection for the particular radar setup (see section 2.2.4). By the idle time value itself velocity resolution could then be adjusted.

In Fig. 3.21(a) we can see that velocity resolution depends on  $T_i$ ,  $T_r$  and overall chirp duration  $T_c$  which is its sum. In Fig. 3.21(b) there is the dependence of MUR on idle time for fixed bandwidth and overall chirp time. Range – Doppler heatmap (RDH) plots addressing adjustment of  $T_i$  are introduced as Fig. B.1 and Fig. B.2.



(a) Velocity resolution as a function of ramp time and idle time. In the plot  $N = 128$  samples of the ramp and starting frequency of 77 GHz has been used.



(b) MUR as a function of idle time for fixed bandwidth of a chirp and fixed overall duration of a chirp.

**Fig. 3.21:** How idle time changes performance of velocity and range estimation.

In [32, p.9] there is a table with minimum  $T_i$  suggestions for various bandwidths of a ramp. It assumes sampling frequency  $f_p$  higher than 5 MHz. We have used the given values as a constraints for our final chirp design.

Ramp bandwidth (GHz)	Minimum idle time ( $\mu\text{s}$ )
Less than 1	2
1 to 2	3.5
2 to 3	5
More than 3	7

**Tab. 3.2:** Times which the frequency synthesizer of the TI AWR1642Boost needs to overcome the gap between the highest and the lowest frequency of a chirp. From [32, p.9], edited.

### 3.3.3 ADC start time

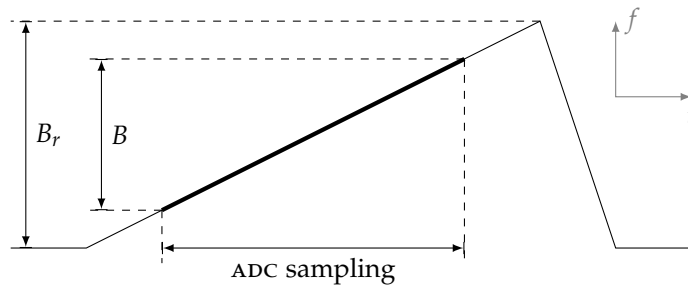
The ADC start time  $T_{\text{ADC}}^s$  has to be tuned so that the whole ADC sampling time  $T_{\text{ADC}}$  fits into the *ramp* part. In terms of the notation from Fig. 3.6:  $T_{\text{ADC}}^s < T_r - T_{\text{ADC}}$ . If the condition is met, the value of  $T_{\text{ADC}}$  has no impact on performance of the range – Doppler matrix.

### 3.3.4 Ramp time

The ramp time  $T_r$  together with frequency slope  $S$  determine *bandwidth*  $B_r$  of a chirp. The  $B_r$  should not be confused with the bandwidth  $B$  from equation (1.14); Bandwidth  $B$  is only a bandwidth of the part of a ramp on which sampling is done and it depends on number of samples taken, sampling frequency, and the slope. Bandwidth  $B_r$  is always greater than  $B$  and is determined by

- bandwidth  $B$  which it has to cover
- limits of the radar transceiver.

The two bandwidths are explained in Fig. 3.22.



**Fig. 3.22:** On the two bandwidth parameters of a chirp. For the timing parameters of a chirp, refer to Fig. 3.6.

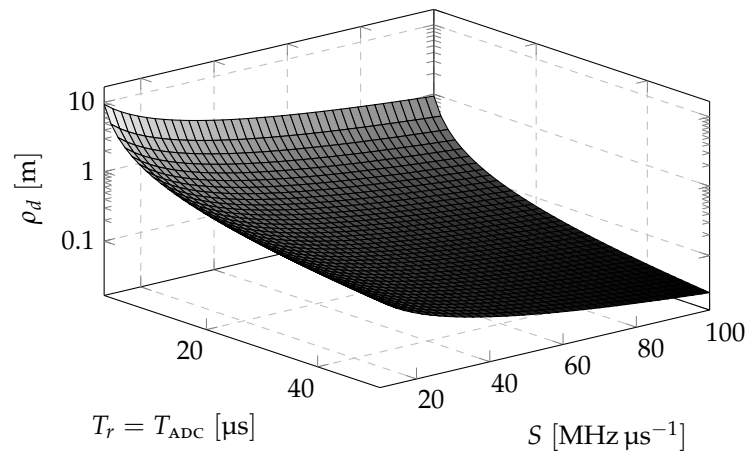
The *ramp time* parameter should be set in a minimal manner: Its value should be *slightly bigger* than  $T_{\text{ADC}} + N/f_p$  with  $N$  being the number of samples taken during one chirp and  $f_p$  being sampling frequency. The reason for setting the parameter *slightly bigger* is a time margin for various delays which could occur in the circuits.

As could be seen from Fig. B.3, the only effect of prolonging ramp time and concurrently leaving the sampling frequency and number of samples constant is a change in  $v_{\text{MAX}}$  and  $\rho_v$ .

### 3.3.5 Frequency slope

Referring to section 1.3.3, neither maximum unambiguous velocity nor velocity resolution is influenced by the frequency slope. Nevertheless, an impact of the slope on MUR (maximum unambiguous range) and range resolution is surely present. The steeper slope is, the lower ramp time is required

for a fixed bandwidth value. If sampling time  $T_{\text{ADC}}$  equals ramp time  $T_r$ , both  $T_r$  and  $S$  influence range resolution and MUR. Dependence of  $\rho_d$  on ramp time and slope is depicted in Fig. 3.23



**Fig. 3.23:** Dependence of range resolution on  $T_r$  and  $S$ , assuming  $T_r = T_{\text{ADC}}$ .

### 3.3.6 Peak to average power ratio in range – Doppler matrix

When a radar, set up as described by Fig. 1.13, is travelling horizontally above conductive ground and no obstacles are present (the radar “sees” only ground), elements of the ground surface perform as elementary reflectors to it. Consequentially, the range – Doppler heatmap contains a “ridge”: a Doppler peak in each range bin corresponds to particular velocity, and the Doppler value of the peak is proper to *all* the range bins (see also Sec. 3.2.1). By examining moments of Doppler peak over range bins, we are able to estimate velocity. Particular algorithms for that are described in section 3.2.

The chirp train has to be designed in order to maximize a (power) gap between the peak and an average Doppler value in all range bins. The ratio between the maximum and average value, averaged over all range bins will be called PAPR (peak to average power ratio). The name should not be confused with PAPR in digital communications [36]. For each configuration, the average PAPR value is indeed different. Maximizing it could be based on measuring it for all possible configurations, and finding an argument (configuration) of its maximum. That, however, is not an acceptable approach since the number of possible configurations is infinite. It is possible to use only some configurations with the values of variables given by an  $N$ -dimensional grid. Nevertheless, from the previous sections it is evident that some configuration variables could have higher impact on the PAPR value, while on some other the PAPR could be independent at all. Each point in the  $N$ -dimensional configuration space (or grid) requires reconfiguring the device. Measuring the average PAPR on an appropriately dense grid would be very time-consuming, and will also include many spare measurements. It could be therefore useful to have a model of PAPR based on *some* of the configuration parameters. Based on it, we could find a region of the configuration parameter space where the PAPR is maximal, and provide the measurements only in that region. In the following two sections, such model would be found.

### 3.3.7 Model 1

A two-parameter model of PAPR has been developed using the following procedure:

1. A *Python* class for automatic radar configuration and PAPR calculation has been written. It could be found on the attached CD in /possible\_configurations/ folder. The basic usage of the class is by calling its member function `measure(self, n_measurements, n_packets,`

filename). Initially, the function opens the given filename in write ('w') mode, and writes a header to the very first line. The header is fixed:

```
1 f1[GHz] Ti[us] Tr[us] S[MHz/us] fs[kHz] papr[-]\n
```

**Listing 3.2:** A fixed header of the file on an output of the measure function.

After writing the header, the function does the following steps repeatedly  $n_{\text{measurements}}$  times:

- i. Selects random chirp train parameters. The random selection does not perturb the whole configuration; it only changes the parameters  $T_i$ ,  $T_r$ ,  $S$ , and  $f_p$  of the profileCfg CLI command of the original firmware. Their values are selected from equally-spaced samples defined by Tab. 3.3. In each iteration and for each variable, an index is generated from uniform distribution  $\mathcal{U}\{0, \text{Number of samples}\}$ . The corresponding value of the variable is then used for configuring the device.

Parameter	Minimum	Maximum	Number of samples
$T_i$	5	500	100
$T_r$	5	500	100
$S$	10	150	100
$f_p$	50	8000	100

**Tab. 3.3:** The grid of parameters for random selection. The values are in units of the profileCfg command – time parameters in  $\mu\text{s}$ , frequency slope in  $\text{MHz } \mu\text{s}^{-1}$ , and sampling frequency in kHz.

- ii. Generates a configuration file with the values from the preceding step, opens the UART port, and configures the radar EVM. If the last command of the configuration file—sensorStart command—is not confirmed (the 'Done' backlog is not sent via UART), step i. is repeated. Otherwise, range-averaged PAPR is computed from the range – Doppler matrix sent in packets. Together with values of the variable parameters from the configuration file, the average value of such PAPR computed from  $n_{\text{packets}}$  subsequent packets is stored in a new line of the file specified by filename.

An example of the final data file filename could be seen in Lst. 3.3.

```
1 f1[GHz] Ti[us] Tr[us] S[MHz/us] fs[kHz] papr[-]
2 77 390.0 65.0 36.87 5269.7 1.2528413
3 77 375.0 40.0 89.19 4707.58 1.3614658
4 77 405.0 150.0 24.14 3422.73 1.212606
...
98 77 145.0 90.0 15.66 2860.61 1.2506488
99 77 355.0 100.0 36.87 4065.15 1.2489502
100 77 20.0 125.0 12.83 5831.82 1.1326853
```

**Listing 3.3:** An example file with measured PAPR for different profileCfg configurations

2. Level of PAPR has been measured for 200 valid configurations with 15 consecutive packets averaged for each of the configurations. Only parameters of the profileCfg command varied; The rest of the configuration file remained unchanged for all measurements:

```
100 sensorStop
100 flushCfg
100 dfeDataOutputMode 1
100 channelCfg 15 3 0
100 adcCfg 2 1
100 adcbufCfg -1 0 0 1 0
100 profileCfg 0 77 370.0 7 50.0 0 0 79.29 1 64 2298.48 0 0 30
```



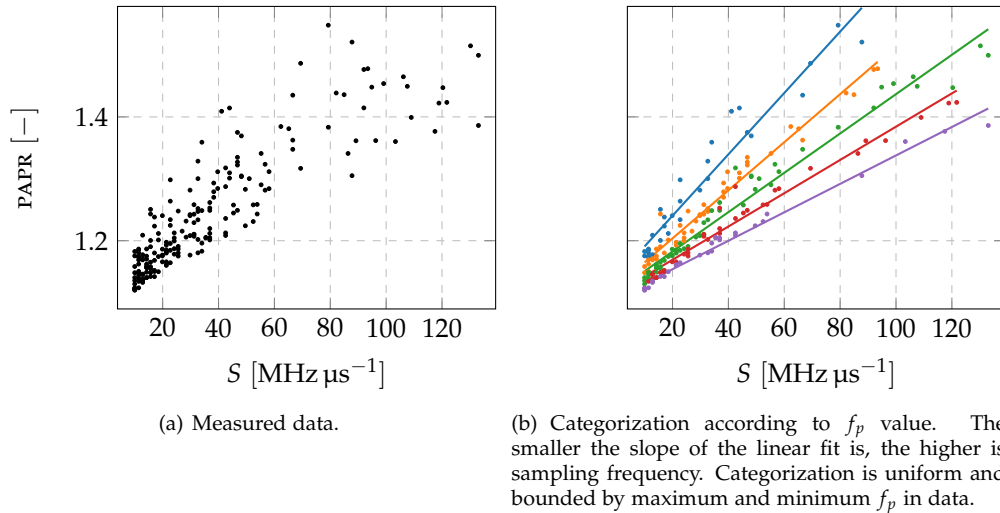
```

100 chirpCfg 0 0 0 0 0 0 0 1
100 chirpCfg 1 1 0 0 0 0 0 2
100 frameCfg 0 1 32 0 250 1 0
100 lowPower 0 1
100 guiMonitor -1 0 0 0 0 1 0
100 cfarCfg -1 0 0 8 4 4 0 5120
100 cfarCfg -1 1 0 4 2 3 0 5120
100 peakGrouping -1 1 1 1 1 1 63
100 multiObjBeamForming -1 1 0.5
100 clutterRemoval -1 0
100 calibDcRangeSig -1 0 -5 8 256
100 extendedMaxVelocity -1 0
100 compRangeBiasAndRxChanPhase 0.0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
100 measureRangeBiasAndRxChanPhase 0 1.5 0.2
100 sensorStart

```

**Listing 3.4:** A complete configuration file for PAPR measurements. Possible configurations varied only in the blue line.

Resulting data have 4 degrees of freedom, excluding the value of PAPR which is to be measured/ modeled. We observed that there is a very strong correlation between PAPR and frequency slope. The result of Spearman's correlation test<sup>vii</sup> is ( $R = 0.90$ ;  $p < 0.01$ ). From Fig. 3.24(a) it is clear that the correlation is caused mainly by HW limitations. Additionally, if we categorize the data according to its sampling frequency value (Fig. 3.24(b)), it is obvious that also  $f_p$  has a strong effect on value of PAPR.



**Fig. 3.24:** Measured dependence of PAPR on  $S$  and  $f_p$ .

The use of  $f_p$  feature as the second parameter of the model is not random. It is based on the correlation matrix of the features (Tab. 3.4). From that, it is clear that PAPR has the strongest correlation with frequency slope  $S$ . The second strongest correlation of PAPR is with ramp time  $T_r$ , however,  $T_r$  is strongly correlated to  $S$ . Since the third most strongly correlated feature, sampling frequency  $f_p$ , is almost uncorrelated to  $S$ , we have selected  $f_p$  as the appropriate second parameter of the model.

A general form of the first attempt to the model has been selected as

$$\zeta(S, f_p) = k(f_p)S + q, \quad (3.6)$$

<sup>vii</sup>Neither the PAPR values nor the values of  $S$  are normally distributed. Results of two-sided Kolmogorov-Smirnov test (normal distribution,  $\mu = 0$ ,  $\sigma = 1$ ) are ( $D = 0.15$ ;  $p < 0.01$ ) and ( $D = 0.17$ ;  $p < 0.01$ ) for PAPR and  $S$ , respectively.



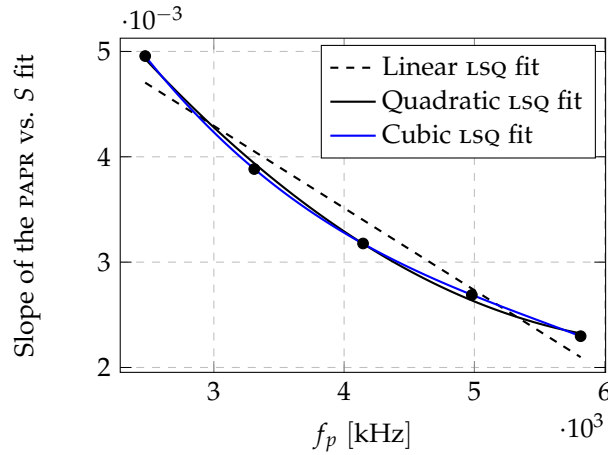
	$T_r$	$S$	$f_p$	PAPR
$T_i$	-0.05	0.06	0.02	0.05
$T_r$		-0.63	-0.05	-0.59
$S$			0.06	0.90
$f_p$				-0.34

**Tab. 3.4:** Correlation between the measured features.

where  $\zeta$  is PAPR,  $k$  depends only on  $f_p$ , and  $q$  is constant. Function  $k(f_p)$  has been approximated by polynomial of third order

$$k(f_p) = af_p^3 + bf_p^2 + cf_p + d \quad (3.7)$$

using the least-squares (LSQ) optimality criterion, with coefficients of the polynomial approximation  $(a, b, c, d) = (-3.79 \cdot 10^{-14}, 6.33 \cdot 10^{-10}, -3.98 \cdot 10^{-6}, 1.15 \cdot 10^{-2})$ .



**Fig. 3.25:** LSQ fits of dependence of fit slope of Fig. 3.24(b) on  $f_p$ .

The  $q$  parameter of (3.6) has been estimated using LSQ again as  $q = 1.1179$ . After calculating all 5 parameters, an error  $\chi$  has been computed as

$$\chi(S, f_p) = \zeta_{\text{measured}}(S, f_p) - \zeta(S, f_p) \quad (3.8)$$

for all available  $(S, f_p)$  pairs. The error values have median very close to zero as could be seen from Fig. 3.26, however, the error distribution is not normal<sup>viii</sup>. Since for a trustworthy model the error distribution *should* be normal, we further inspected how the  $\chi$  values are correlated with the measured quantities. The results are in Tab. 3.5. The results have shown that the error was still highly correlated to sampling frequency  $f_p$

The error has been approximated by a superposition of an exponential function and a linear function with parameters  $A, B, C, D$ , and  $E$

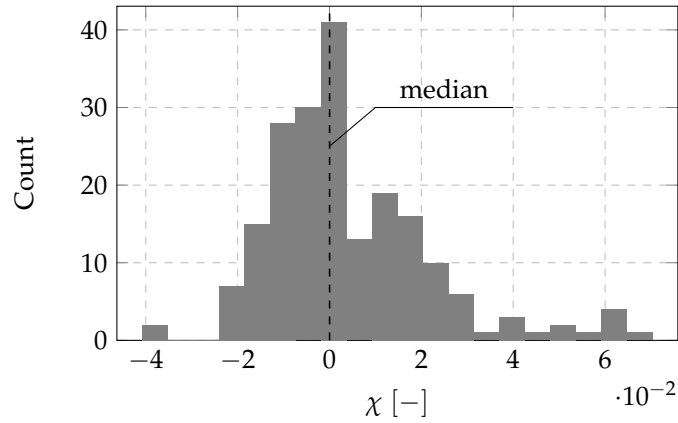
$$\chi(f_p) \propto A \exp((f_p - E)B) + C(f_p - E) + D. \quad (3.9)$$

For  $\chi$  in its 95% confidence interval, LSQ fit has resulted in  $(A, B, C, D, E) = (5.008 \cdot 10^{-2}, -3.645 \cdot 10^{-4}, -1.00 \cdot 10^{-6}, -1.431 \cdot 10^{-2}, 1.418 \cdot 10^3)$ . Error and its fit is depicted in Fig. 3.27.

An error of the corrected model

$$\begin{aligned} \zeta_c(S, f_p) &= k(f_p)S + q + \chi(f_p) = \\ &= (af_p^3 + bf_p^2 + cf_p + d)S + q + A \exp((f_p - E)B) + C(f_p - E) + D \end{aligned} \quad (3.10)$$

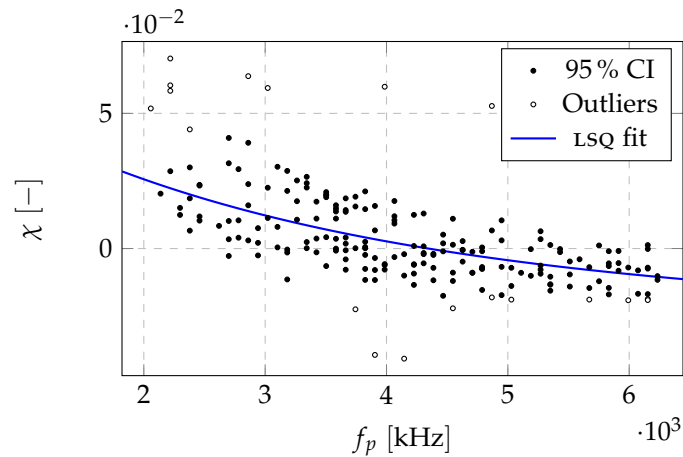
<sup>viii</sup>Shapiro-Wilk test results in  $(W = 0.92; p < 0.01)$ .



**Fig. 3.26:** Histogram of error from (3.8).

Measure	Correlation
$T_i$	( $R = -0.03$ ; $p = 0.72$ )
$T_r$	( $R = -0.11$ ; $p = 0.12$ )
$f_p$	( $R = -0.67$ ; $p < 0.01$ )
$S$	( $R = 0.23$ ; $p < 0.01$ )

**Tab. 3.5:** Correlation between model error  $\chi$  and the measured quantities. Spearman's correlation test has been used.

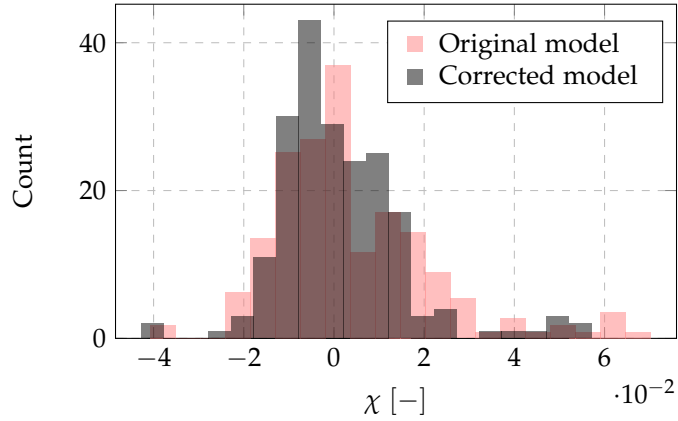


**Fig. 3.27:** LSQ fit of model error  $\chi$  by function (3.9).

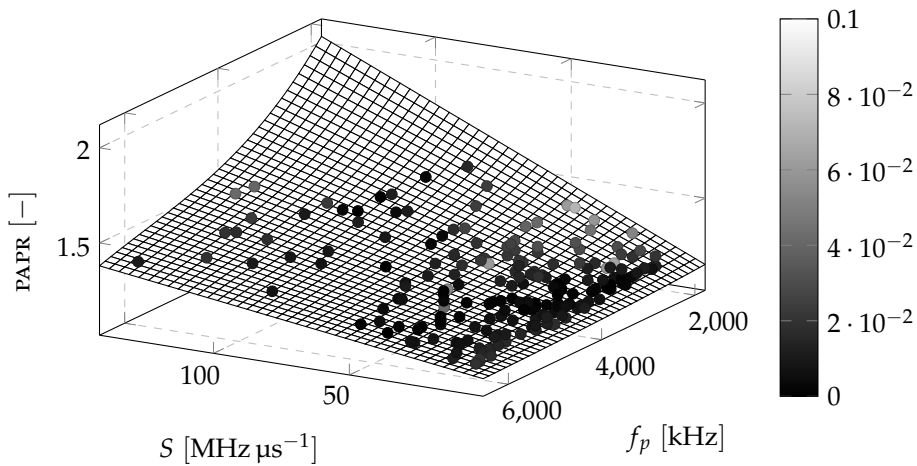
with parameters  $(a, \dots, d, A, \dots, E)$  has distribution much more symmetric around zero than the error of the original model, which could be seen from histograms in Fig. 3.28. Even though the error distribution is not normal, we consider the absence of significant bias a property that is sufficient for the model. The corrected model (3.10) and the measured data are plotted together in Fig. 3.29.

### 3.3.8 Model 2

The model from Section 3.3.7 is simple and mirrors the trend in data well. Nevertheless, its derivation is rather *ad hoc* and there has been a vast chance that a more systematic approach



**Fig. 3.28:** Histograms of error  $\chi$  of the original model (3.6) and the corrected model (3.10).



**Fig. 3.29:** The measured data together with the model of them. The color of the data points expresses an absolute value of error  $\chi$ .

to modeling would accomplish more accurate model. For that purpose, we have used *Python's* *scikit-learn* module [37].

Two linear regression models  $\zeta_{2,1}$  and  $\zeta_{2,2}$  with polynomial kernels have been used: The first one have had a kernel of second order ( $\deg(\zeta_{2,1}) = 2$ ), the second one have had that of third order ( $\deg(\zeta_{2,2}) = 3$ ). General form of the two-variable linear model with polynomial kernel is

$$\zeta_{2,i} = \sum_{m=0}^{\deg(\zeta_{2,i})} \sum_{n=0}^{\deg(\zeta_{2,i})} c_{m,n} x_1^m x_2^n, \quad (3.11)$$

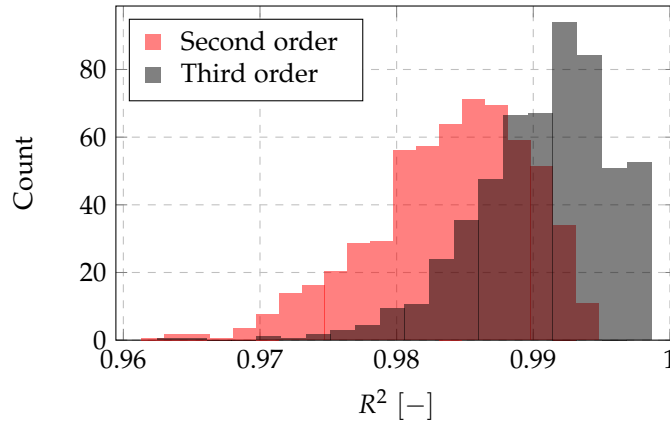
where  $x_1$  and  $x_2$  are the two parameters. We have used the same pair of parameters— $S$  and  $f_p$ —which has the same reason as described in the previous section. The general model (3.11) with our variables substituted in is

$$\zeta_{2,i} = \sum_{m=0}^{\deg(\zeta_{2,i})} \sum_{n=0}^{\deg(\zeta_{2,i})} c_{m,n} S^m f_p^n. \quad (3.12)$$

Both variants of the model were trained and cross-validated using *Monte Carlo* cross-validation with

- 70% of data for training, 30% of data for testing.
- $R^2$  (Coefficient of determination) as a regression score function.

Goodness of fit turned out to be better with the third order 2D polynomial used as a kernel, which could be seen from histograms in Fig. 3.30. The distribution of  $R^2$  values also indicates that the model is not overtrained.



**Fig. 3.30:** Histograms of Coefficient of determination  $R^2$  for linear models with second order and third order polynomial kernels.

The coefficients of the third-order polynomial model with  $R^2 \sim$  (Median = 0.991, IQR = 0.994 – 0.988) are listed in Tab. 3.6. The error  $\chi$  (3.8) within its 95% confidence interval has normal distribution ( $\alpha = 0.05$ ) with Kolmogorov-Smirnov test result ( $D = 0.096$ ;  $p = 0.056$ ).

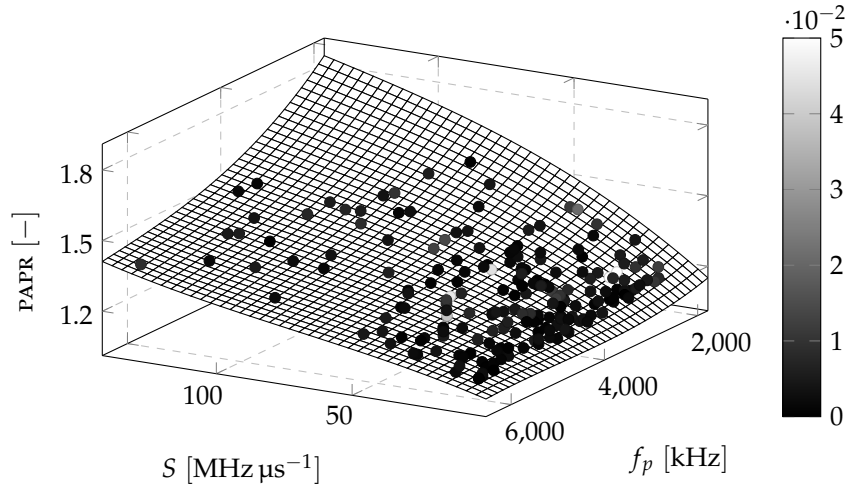
Coefficient	Value
$c_{0,0}$	1.346
$c_{0,1}$	$-16.758 \cdot 10^{-5}$
$c_{0,2}$	$3.489 \cdot 10^{-8}$
$c_{0,3}$	$-2.353 \cdot 10^{-12}$
$c_{1,0}$	$11.5 \cdot 10^{-3}$
$c_{1,1}$	$-1.842 \cdot 10^{-6}$
$c_{1,2}$	$8.627 \cdot 10^{-11}$
$c_{2,0}$	$-4.488 \cdot 10^{-5}$
$c_{2,1}$	$4.131 \cdot 10^{-9}$
$c_{3,0}$	$8.407 \cdot 10^{-8}$

**Tab. 3.6:** Coefficients of the linear model of PAPR with third-order polynomial kernel (3.12).

The modeled surface and measured data are depicted in Fig. 3.31.

### 3.3.9 Selecting the best configuration

For all the algorithms proposed above, a configuration parameter most important for ensuring consistent and accurate velocity estimation is its *average PAPR value*. In sections 3.3.7 and 3.3.8 we have modeled PAPR using two-parameter models employing *frequency slope* and *sampling frequency*. Since the model from Sec. 3.3.8 has exhibited better coherence with observed data, we have selected it as the one with which we work further.



**Fig. 3.31:** The measured data together with predictions of the linear model with third-order polynomial kernel. The color of the data points expresses an absolute value of error  $\chi$ .

Straightforward unconstrained adjustment of the parameters aiming to higher PAPR values is not possible because of HW limitations. The constraints we have to address are those from the following list:

#### Constraints related to $f_p$ only.

1. Maximum possible sampling frequency of the digital front-end (DFE). Its maximum value is  $18.75 \text{ MHz s}^{-1}$  for complex sampling mode [22,31]. Nevertheless, this constraint is not relevant for us because—as the model reveals—we need to use as low  $f_p$  as possible to maximize PAPR.
2. Minimum possible sampling frequency of the DFE. The original firmware does not allow for sampling rates lower than 2000 kHz [38].

#### Constraints related to both $f_p$ and $S$ .

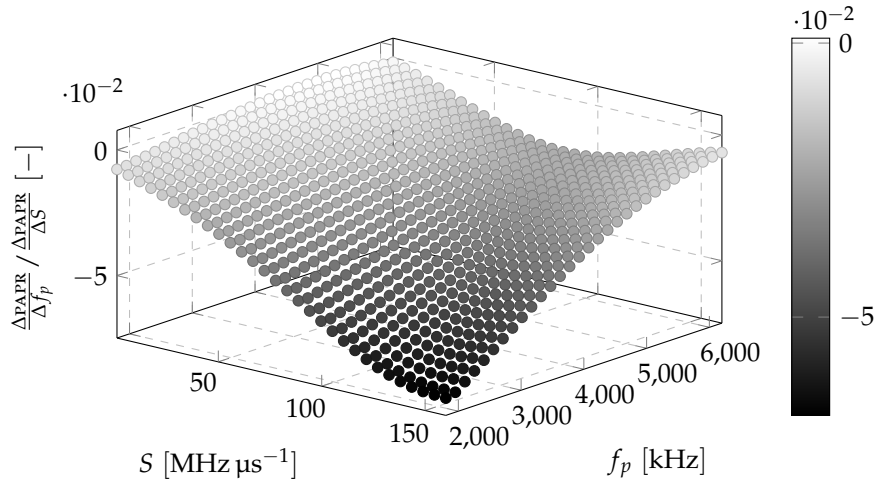
1. Frequency range of the EVM. As has been mentioned above, the TI AWR1642BOOST could only use chirps occupying frequencies from 77 GHz to 81 GHz. Taking into account that it takes some time to gather the required number of samples during the chirp ramp, and that the sampling could not start *right after* a ramp start, we have to manage for slope  $S$  *small enough* not to reach the upper frequency limit 81 GHz. From the perspective of this constraint it is also advantageous to use as low start frequency of a chirp as possible (77 GHz).
2. Minimum number of samples for each chirp. With sample quantity, granularity of a range – Doppler matrix in the range axis raises, which we could help the velocity evaluation algorithms perform better. Referring to Sec. 3.1.2, the number has to be a multiple of 4 bigger or equal to 64.

Besides the constraints, we also need to take into account a gradient of PAPR. At each point of the  $(S, f_p)$ -plane, that makes a difference in PAPR increase or decrease whether we shift  $S$ , or  $f_p$ . Differences in both  $S$  and  $f_p$  directions have been computed from PAPR model and a ratio

$$\frac{\left. \frac{\Delta \text{PAPR}}{\Delta f_p} \right|_{\Delta f_p = 1 \text{ kHz}}}{\left. \frac{\Delta \text{PAPR}}{\Delta S} \right|_{\Delta S = 1 \text{ MHz } \mu\text{s}^{-1}}} \quad (3.13)$$

has been expressed from them. The ratio (3.13) is plotted in Fig. 3.32. A conclusion induced by the values of it could be

- For a region of small slopes and big sampling frequencies, there is almost no difference in PAPR shift between shift of slope by  $1 \text{ MHz } \mu\text{s}^{-1}$  and shift of sampling frequency by 1 kHz.
- For a region of big slopes and small sampling frequencies, a change of  $f_p$  by 1 kHz has much higher impact on PAPR than a change of  $S$  by  $1 \text{ MHz } \mu\text{s}^{-1}$ .



**Fig. 3.32:** Scatter plot of ratio between a change in PAPR when  $f_p$  is shifted by 1 kHz and a change in PAPR when  $S$  is shifted by  $1 \text{ MHz } \mu\text{s}^{-1}$  for each point of the  $(S, f_p)$ -plane. The second measure is always considered constant.

A program has been written that finds out whether a configuration is possible for all points of the given  $(S, f_p)$ -grid. The program uses the following `profileCfg` parameter values and dependencies:

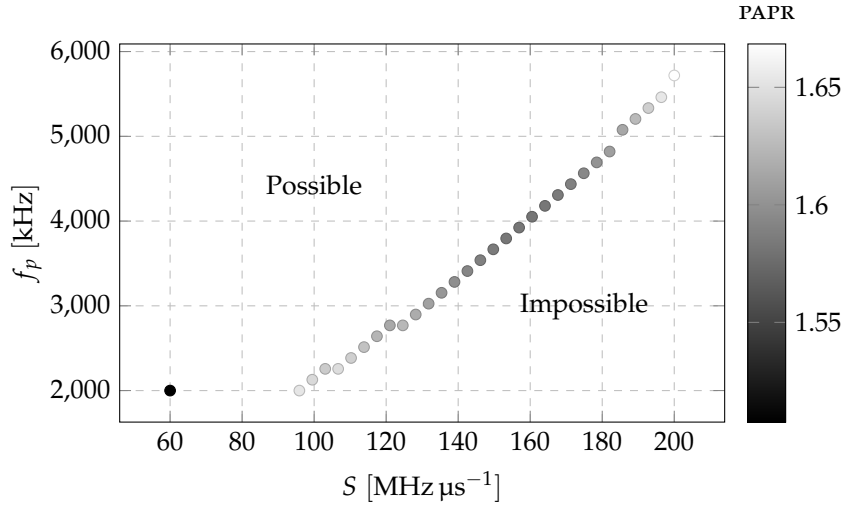
Parameter	Value
$f_1$	77 GHz
$N_{\text{FFT}}$	64
$T_{\text{ADC},S}$	7 $\mu\text{s}$
$T_r$	$1.1(N_{\text{FFT}}/f_p + T_{\text{ADC},S})$
$T_i$	7 $\mu\text{s}$

**Tab. 3.7:** `profileCfg` parameter values and dependencies employed in the program for detecting possible configurations.

From given limits and quantity for both  $S$  and  $f_p$ , it generates a grid. Then, it point-by-point changes the `profileCfg` line of a configuration file and catches a response for the last command of the file (`sensorStart`). The response is “Done.” for successful configuration, and “Error -1” for unsuccessful. As an output, it plots a scatter plot of the result ( $x$  and  $y$  coordinates  $S$  and  $f_p$ , respectively,  $z$  coordinate 1 for *possible*, 0 for *impossible*). The program is available on the attached CD as `config_finder.py` in the `/possible_configurations/` folder.

The program has been run for the first time on a wide grid of configurations: Range for  $S$  had been set to  $(60, 200, \text{num} = 40)^{\text{ix}}$ , and that for  $f_p$  to  $(2000, 7000, \text{num} = 40)$ . The measured boundary between *possible* and *impossible* configurations is plotted in Fig. 3.33.

<sup>ix</sup>Notation  $(a, b, \text{num} = n)$  stands for a uniformly spaced axis of  $n$  values from  $a$  to  $b$ , both inclusively.



**Fig. 3.33:** The boundary of possible configurations in  $(S, f_p)$ -plane. All the points plotted represent *possible* configurations. Color of the points display the modeled PAPR value.

As Fig. 3.33 reveals, there are two optimal configurations with respect to PAPR level:

1.  $S = 92.3 \text{ MHz } \mu\text{s}^{-1}$ ,  $f_p = 2000 \text{ MHz } \mu\text{s}^{-1}$ , other `profileCfg` parameters from Tab. 3.7.
2.  $S = 200 \text{ MHz } \mu\text{s}^{-1}$ ,  $f_p = 5717.95 \text{ MHz } \mu\text{s}^{-1}$ , other `profileCfg` parameters from Tab. 3.7.

Since the second option should have higher power consumption due to more dense sampling and steeper ramp, we decided to use the first option as the right configuration for the final application. Moreover, we have the luck that the range of ranges (0 m to 3.25 m) and the range of velocities ( $-9.76 \text{ m s}^{-1}$  to  $9.76 \text{ m s}^{-1}$ ) is perfect for our application.

### 3.4 Velocity evaluation

The knowledge gathered in the previous three sections could be summarized in these two points:

- The quantity best reflecting horizontal velocity changes out of the tested ones is *mean of Doppler bins argmax* ( $\mu_D$ ).
- Configuration with `profileCfg` parameters set as Tab. 3.7 offers the best possible PAPR.

Those two determine a method and a radar configuration we should use in order to measure horizontal velocity as accurately as possible. Nevertheless, the dependence between  $\mu_D$  and horizontal velocity  $v_x$  is unknown and has to be found.

#### 3.4.1 Measurement methodology and setup

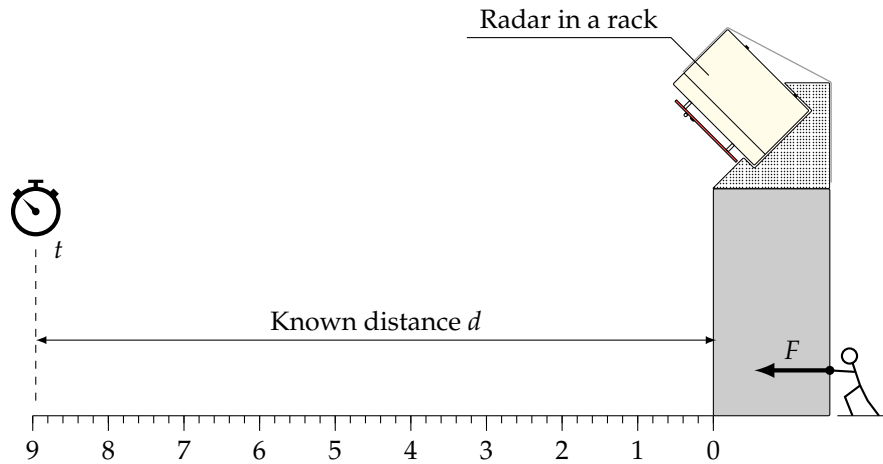
The approach we employed to find the dependence of  $v_x$  on  $\mu_D$  has followed the following steps:

1. Placing the radar tilted by a known angle  $\alpha$  to a constant elevation  $h$  above ground.
2. Managing it to travel a known distance  $d$  with possibly *constant velocity* and measuring the time spent,  $t$ .
3. (*Together with 2.*) Measuring the mean of Doppler bin `argmax`  $\mu_D$  over all range bins of all RDMS received during the travel.



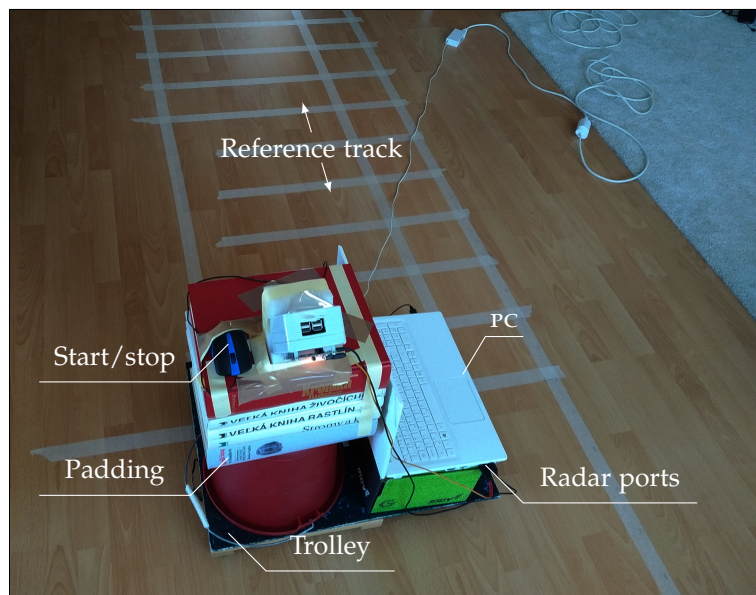
4. Examining the measured data and modeling the dependence of  $(d/t)$  on  $\mu_D$ .

An illustration of the method is in Fig. 3.34.



**Fig. 3.34:** A setup for measuring dependence of  $v_x$  on  $\mu_D$ .

To fulfill the preceding steps we have built a measuring trolley equipped with a padding of known height, on the top of which the radar in a rack has been placed. The rack held the radar in the particular inclination relative to the ground plane. Besides the padding, also a PC has been placed on the top of the trolley. The radar has been powered up from the PC, to which also the data and configuration ports have been connected. The last part of the measuring trolley has been a mouse which left and right buttons controlled a measuring script running on the PC. The trolley and a reference track which it has been pushed along are depicted in Fig. 3.35.



**Fig. 3.35:** The measuring rack with the radar, PC and the mouse.

The track has been designed with respect to a shape of the area of reflection (see Sec. 2.2.4). The padding on the trolley allowed 5 different elevations ranging from 48.5 cm to 63.3 cm.



Referring to Sec. 2.2.3, independently on the angle  $\alpha$ , for all of them ground plane is far enough to allow EVM's antennas work in their far field. The rack holding the radar has been built for fixed angle approximately  $\alpha \approx 45^\circ$ , however, using an additional padding under its back side we were able to measure with angles  $\alpha$  up to  $75^\circ$ . The most area-demanding setup out of the available has been the setup with  $h = 0.633$  m and  $\alpha \approx 45^\circ$ . Such constellation requires 1.6 m of free ground in the front of radar, and 1.714 m of total (both the left and the right side together) free ground in the direction perpendicular to the direction of motion. Angles  $\alpha$  greater than  $75^\circ$  are not realizable for the full range of used elevations; The area of reflection for such angles and  $h = 0.633$  m intersects with an area covered by the trolley itself.

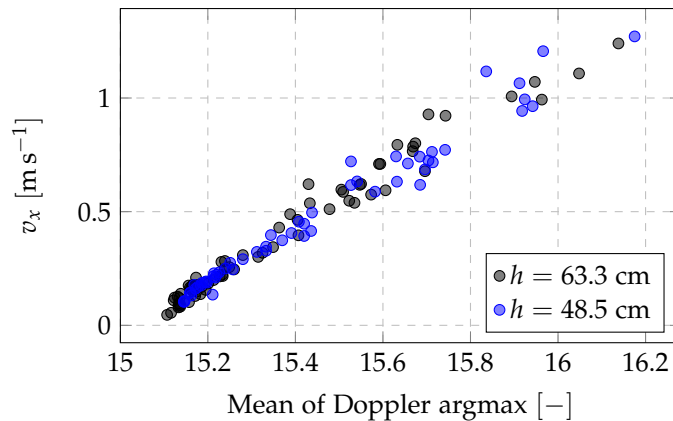
The measuring script is attached as `cd/meas_scripts/meas_trolley.py`. When a measurement is started by clicking the left mouse button, the script stores the time, resets a counter (`n_means`) to zero, empties a list `means`, and enters an infinite loop. In the loop, these steps are repeated:

1. A new packet is read.
2. Its RDM data are rearranged using the scheme from Fig. 4.2.
3. A mean is calculated out of all arguments of Doppler maxima and this mean is appended to the `means` list.
4. The counter `n_means` is incremented.

The program leaves the loop when an interrupt is generated by clicking the right mouse button. In such case, the stop time is stored immediately. Then, the time difference  $t$  is calculated and the known distance  $d$  is divided by it, which results in a velocity estimate  $v_x$ . The means list is summed and divided by `n_means` to obtain the overall mean of Doppler bins argmax. Eventually, a new line is appended to the given file (`results_file`); it contains space-delimited values of  $h$  in meters,  $\alpha$  in degrees,  $v_x$  in meters per second, and mean of Doppler bins argmax (unitless).

### 3.4.2 Analysis of measured data

**Effect of elevation  $h$**  Two samples with equal  $\alpha$  and elevations  $h_1 = 0.633$  m, and  $h_2 = 0.485$  m (the highest and the lowest of the elevations used), respectively were selected for observing an effect of  $h$  on  $v_x$ -on- $\mu_D$  dependence. The samples are plotted in Fig. 3.36.



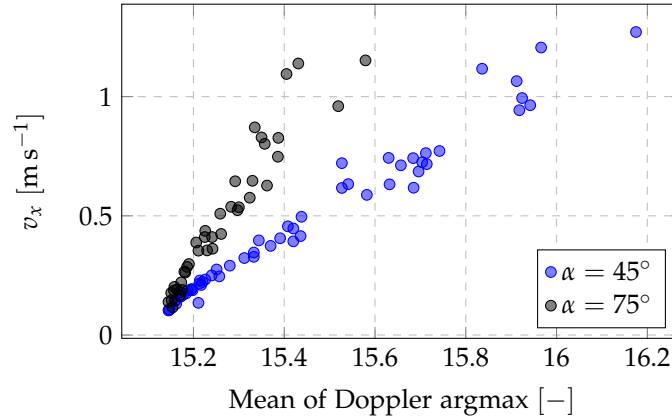
**Fig. 3.36:** Measured data for  $\alpha = 45^\circ$  and two elevations  $h$ .

Both  $\mu_D$  and  $v_x$  data of the two samples have been tested for their competence to the same statistical distribution. The *Mann – Whitney U test*<sup>x</sup> [39, 40] revealed that the data could be

<sup>x</sup>The samples are non-parametric in both dimensions.

considered being competent to the same bivariate distribution with 5% significance level (Result for  $\mu_D$  ( $U = 1564.0$ ;  $p = 0.052$ ), for  $v_x$  ( $U = 1621.5$ ;  $p = 0.10$ )). According to the result we could proclaim that quality of the horizontal velocity evaluation *does not depend on elevation of the radar* if the elevation above the ground plane allows the radar antennas operate in the far field.

**Effect of depression angle  $\alpha$**  Two samples with equal elevation  $h = 0.485$  m and depression angles  $\alpha_1 = 45^\circ$ , and  $\alpha_2 = 75^\circ$  were tested for an effect of  $\alpha$  on  $v_x$ -on- $\mu_D$  dependence. The samples are plotted in Fig. 3.37.



**Fig. 3.37:** Measured data for  $h = 48.5$  cm and two depression angles of the radar.

It is obvious that the  $\mu_D$  values of the two samples do not originate from the same distribution.<sup>xi</sup> Here, more relevant information is whether the distribution of  $v_x$  values between the samples is equal. Taking  $v_x$  as independent variable and  $\mu_D$  as dependent one and performing the MWUT, we obtain ( $U = 1007.5$ ;  $p = 0.52$ ) so the measurements for  $\alpha_1$  and  $\alpha_2$  have been successful in terms of covering the same horizontal velocity values.

Without any further analysis, we can conclude that with larger  $\alpha$ , sensitivity of  $\mu_D$  on  $v_x$  decreases. On the other hand, the larger  $\alpha$  requires smaller area of reflection which could be handy in real usecase. We adjudged  $\alpha = 45^\circ$  being an angle providing good sensitivity with concurrently requiring acceptable area of reflection. Therefore, we will use this depression angle for the final radar device.

**Modeling  $v_x$**  As has been found out in the paragraph *Effect of elevation  $h$* , it is possible to use the measured data as one sample independently on  $h$  value. This way, we dispose with a sample of total 126 entries addressing  $\mu_D$  and  $v_x$  for  $\alpha \approx 45^\circ$  and various elevations from 0.485 m to 0.633 m above a ground plane.

The data clearly manifest heteroscedascity. The heteroscedascity could be considered *pure*: For higher  $v_x$ , the human factor causes bigger errors. Pressing a mouse button *right* on the start line of the track is easier than pressing it at the right moment when the trolley has larger velocity, just the way it is with *stopping* the measuring script by clicking the mouse button when the trolley crosses the finish line of the track. We consider these human-caused errors to be normally distributed.

The data in Fig. 3.36 at first glance exhibit linear dependence, which comes hand-by-hand with our expectations. According to (1.22) the relation between  $v_x$  and the phase shift of the IF signal really *is* linear. It is therefore appropriate to model  $\mu_D$  as a first-order polynomial

$$\mu_D = kv_x + q \quad (3.14)$$

<sup>xi</sup>Rigorously, the result of Mann – Whitney  $U$  test (MWUT) is ( $U = 709.5$ ;  $p < 0.05$ ).

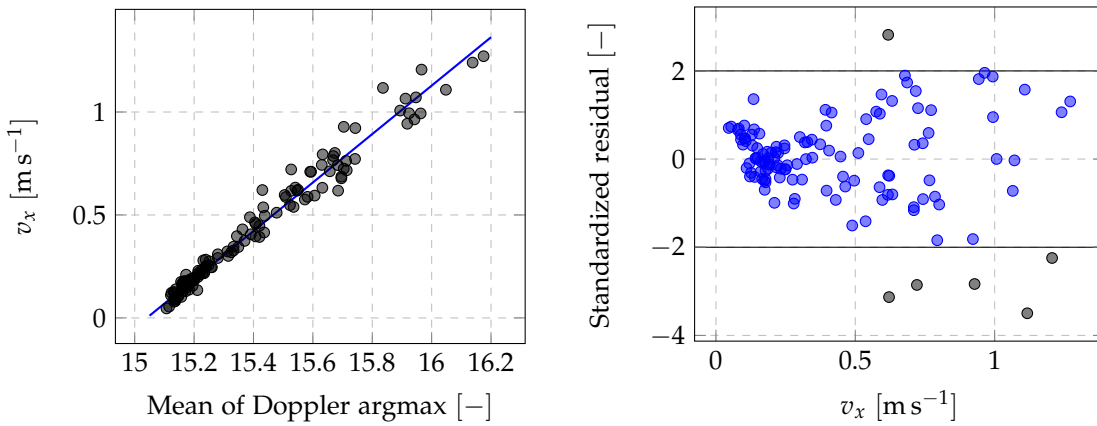
with  $k$  and  $q$  being *slope* and *intercept* of the regression line, respectively. Because of heteroscedascity we cannot use the standard least-squares regression. Nevertheless, the weighted LSQ regression could be used. Assumed relative variances being inverted values of  $v_x$ , we obtain

$$\mu_D = 0.85v_x + 15.035. \quad (3.15)$$

Expressing  $v_x$  from (3.15), we end up with the final linear model of  $v_x$

$$v_x = 1.176\mu_D - 17.688. \quad (3.16)$$

Measured data with model are shown in Fig. 3.38(a). Standardized residuals of the model are depicted in Fig. 3.38(b).



(a) Measured data and their linear model.

(b) Standardized residuals of data for model (3.16). Data within the 95% confidence interval are marked by blue.

**Fig. 3.38:** The final model of  $v_x$  with parameter  $\mu_D$ .

### 3.4.3 Procedure of velocity evaluation

Having the model (3.16), evaluation of velocity from UART data is straightforward. To summarize the steps, we explicitly describe the modus operandi that the UART data undergo on their path from when they come into the second-order processing PC to the final velocity estimate:

1. Extraction of the *range – Doppler matrix* message from the packet.
2. Performing the rearrangements on the RDM itself.
3. Computing  $\mu_D$  of the rearranged RDM.
4. Converting the  $\mu_D$  to horizontal velocity  $v_x$  using (3.16).

### 3.4.4 Quantifying accuracy and precision

The approach from the previous section has been tested for its precision using four scenerios. In all of them, horizontal velocity has been measured by radar and by some other (alternative) method. An effort had been put on selecting the alternative method as accurate as possible since to quantify accuracy of a new measurement method, we need to know the true velocity – or, loosely speaking, *true enough*. Four test were performed in total: The firts three used measuring time spent on a track of a known length together with constant velocity on the track as a velocity reference. The last one used GPS velocity estimate.

**Indoor trolley testing** used a setup equal to that used during measurements of  $v_x$ -on- $\mu_D$  dependence parameters. The trolley from Fig. 3.35 has been used with  $h = 0.485$  m and  $\alpha \approx 45^\circ$ . The track has had the same length  $d = 2$  m. The measurement script used is attached as `CD/meas_scripts/meas_trolley_testing.py`. It have employed the same `tkinter`-based interface for starting and terminating a measurement as the dependence-finding script but the main procedure has been different. In this case it have consisted of the following steps:

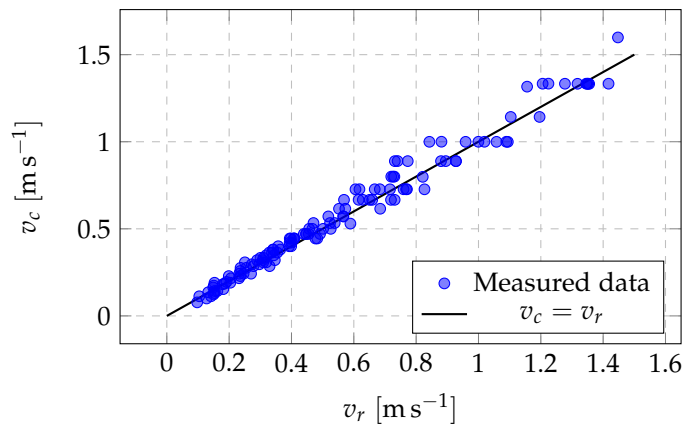
1. Obtaining the RDM from `UART` and rearranging it.
2. (Together with 1.) Storing the time of its arrival.
3. Calculating  $v_x$  based on  $\mu_D$  and (3.16).
4. Appending the results to a file, incrementing a measurement number counter.

In rows of the result file, there are three features stored, namely radar velocity estimate in meters per second, time of packet reception in seconds since epoch, and measurement number. During a measurement, a person pushed the trolley along the track, trying to keep velocity *constant*. Because an operator could be systematically too early or too late in starting or terminating the measurement by a mouseclick, the dataset contains measurements performed by two operators. Competence of a particular measurement to the operators has been added to the dataset post hoc.

For each of the measurements, an average velocity has been calculated as

$$v_c = \frac{t_l - t_f}{d}, \quad (3.17)$$

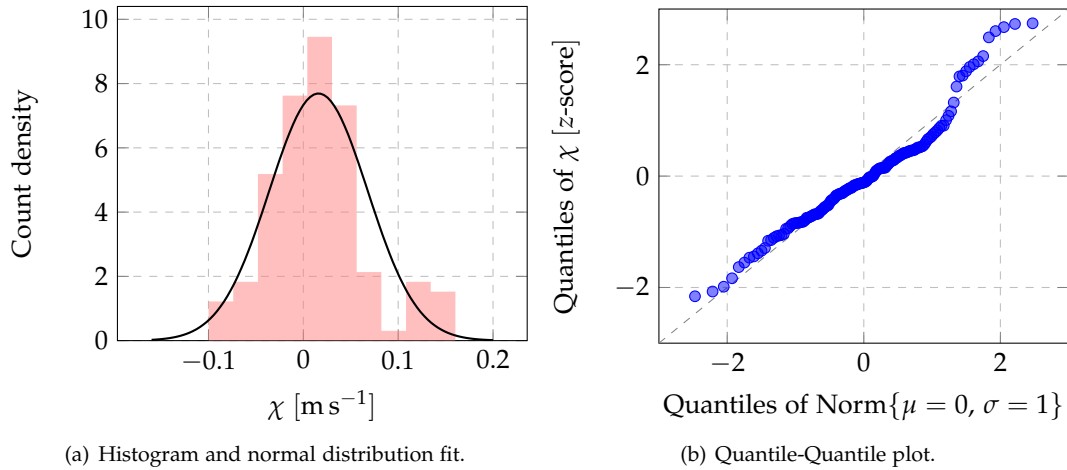
where  $t_l$  and  $t_f$  are time of the last and the first packet within a measurement. For each measurement, mean radar velocity  $v_r$  has been calculated. The measured relationship between  $v_c$  and  $v_r$  is plotted in Fig. 3.39.



**Fig. 3.39:** Mean radar velocity  $v_r$  and calculated average velocity during a measurement  $v_c$ .

Error  $\chi = v_c - v_r$  is expected to have normal distribution with zero mean. Its histogram and Q-Q plot are in Fig. 3.40.

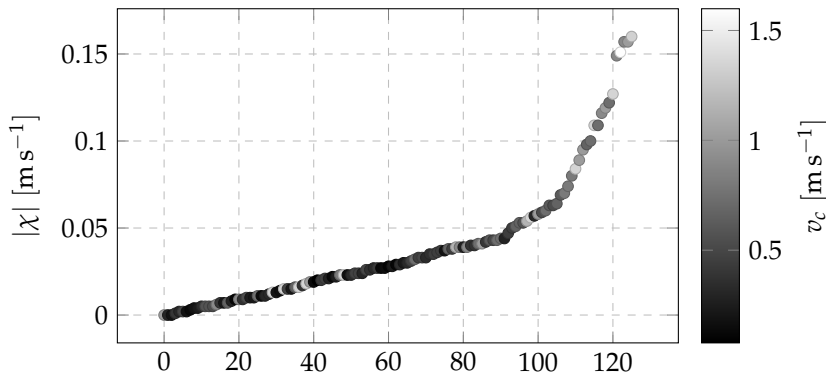
From both the plots in Fig. 3.40(a) and Fig. 3.40(b), it is obvious that normality of the error is only disturbed by the biggest error values. When an absolute value of the error is ordered increasingly, plotted value-by-value (sequence number on  $x$  axis), and colored according to  $v_c$  (the plot is in Fig. 3.41), it could be noted that the biggest error values are mostly connected with higher velocity. Such behaviour is easy to explain: It is again caused by the inaccurately timed mouseclicks for starting and terminating the measuring script when the trolley crossed the start line or finish line. For  $v_c$  bounded by  $v_{c,\max} = 0.885$   $\text{m s}^{-1}$ , error  $\chi$  has normal distribution



**Fig. 3.40:** Distribution of error  $\chi = v_c - v_r$ .

(Shapiro-Wilk test: ( $W = 0.99$ ;  $p = 0.73$ )) with least-squares-fit result ( $\mu = 0.012$ ,  $\sigma = 0.038$ ). The results yield the following conclusions for measuring velocities below  $0.885 \text{ m s}^{-1}$ :

1. Radar measurement  $v_r$  is *slightly biased* (by  $0.012 \text{ m s}^{-1}$ ).
2. Standard deviation of the measurement using the model from (3.16) is  $0.038 \text{ m s}^{-1}$ .

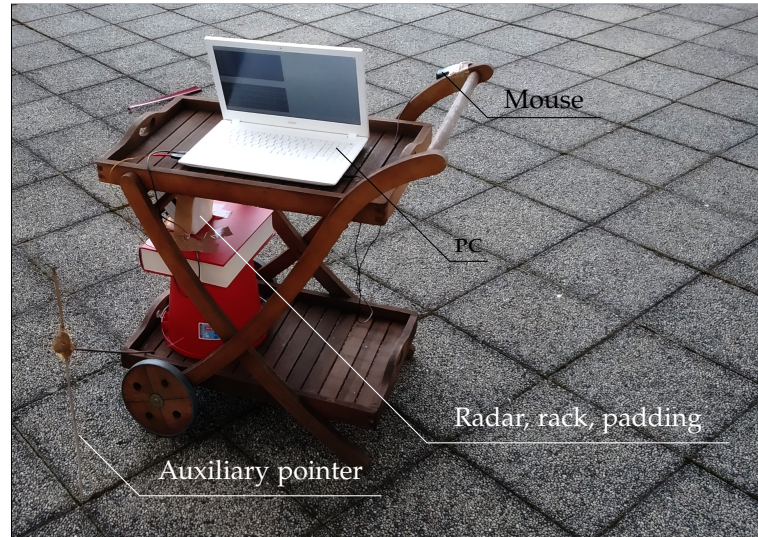


**Fig. 3.41:** Absolute values of error colored by  $v_c$ .

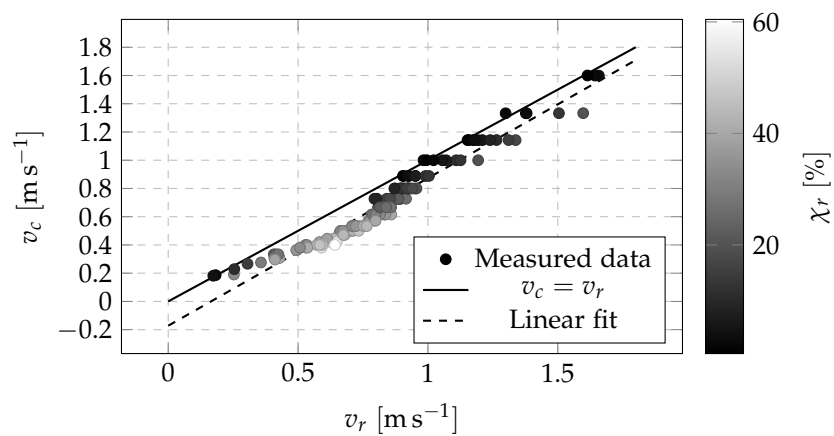
**Outdoor trolley testing** was provided with a goal to see how the radar performs in challenging, very bumpy environment. A setup was very similar to that for indoor trolley testing, however, another trolley and another surface were employed. The full setup is in Fig. 3.42.

The garden trolley used has had wheels without bearings and on its backside, there even have not been any wheels. The ground plane has been made of concrete tiles covered by small stones, each of them approximately 4 mm in diameter. The size of the stones has therefore been similar to wavelength of the radar. An auxiliary pointer was mounted on the trolley to help an operator recognize the moments of crossing the start line and the finish line of the track. The track was 201 cm long. The measurements were gathered in temperature approximately  $5^\circ\text{C}$ .

The measuring script was the same as for the indoor measurement, and so was the methodology. The measured  $v_c$  vs.  $v_r$  plot is in Fig. 3.43.



**Fig. 3.42:** Setup for outdoor measurements on bumpy ground.

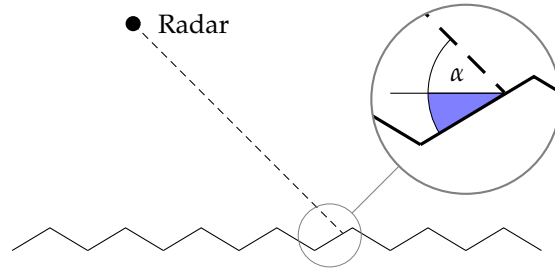


**Fig. 3.43:** Mean radar velocity  $v_r$  and calculated average velocity during a measurement  $v_c$ . Color of the points is determined by an error  $\chi_r = |v_c - v_r|/v_c$ .

From the plot we can see the velocity  $v_r$  is systematically *underestimated*. For the medium velocities, the relative error reaches more than 50% which is not acceptable. Two explanations of such behaviour are:

1. The shape and dimensions of the stones covering the tiles could have caused an “angle enlargement” effect. Refer to Fig. 3.44 for an explanation. The profile of the the stones could make the incident angle  $\alpha$  larger from the radar’s point of view. Referring to Fig. 3.37, the enlargement of  $\alpha$  causes a decrease of  $\mu_D$  that—assuming the true value of  $\alpha$  being  $45^\circ$ —reversely causes a decrease in estimated velocity value.





**Fig. 3.44:** On the angle enlargement effect.

Such viewpoint gives a birth to a question why the opposite sides of the obstacles do not compensate the error by their analogous “angle reduction” effect. The answer could be in the particular arrangement of the surface. The stones have dimensions that cause the surface being *rough* according to *Rayleigh criterion* for surface roughness [19,41]

$$\text{Surface is rough} \Leftrightarrow h > \frac{\lambda}{8 \sin(\alpha)} \quad (3.18)$$

where  $h$  is the “typical” distance between two points on the surface where the wave is reflected measured on the line parallel with the wave (the criterion is based on a condition for maximum allowed phase difference for two waves reflected on the surface originating from the same incoming wave). If the surface is rough, reflection is generally considered *diffuse*. That means, the reflected wave is not directional, but rather diffused to all possible directions. The combination of angle enlargement on one hand, and the diffuse reflection on the other hand could have caused the observed errors.

2. An impact of *temperature* on the radar performance. Since the measurements for deriving (3.16) were taken in an indoor environment with temperature 22 °C and the temperature outdoors during the outdoor trolley measurements had not exceed 5 °C, it is relevant to suspect the temperature difference of causing the observed errors. Nevertheless, the measurement was repeated in warmer outside conditions with no significant difference. Therefore, we cannot yield that the temperature have caused the errors.

**Indoor car measurement.** Another approach to precision testing have used a car driving in a big garage. The surface of the ground plane has been very smooth – the ground has been made of glazed concrete. The experimental setup is schematically drawn in Fig. 3.45. The radar has been fixed on the trunk rim, with the pc and its operator inside the trunk. A driver have driven the car with possibly constant velocity in both directions (drive forward, reverse drive). The operator used a pointer and the mouse to start and terminate the measuring script right when the car crossed indication lines of the 8 m long track. The measuring script have had the same structure as with the preceding approaches. 28 records have been taken at total, from that 14 for drive forward (radar have measured *negative* velocity), and 14 for reverse drive (radar have measured *positive* velocity).

From the records, four were identified as outliers and discarded based on their error  $\chi = v_c - v_r$  and criterion

$$\chi \in \langle Q_1 - 1.5\Delta_Q; Q_3 + 1.5\Delta_Q \rangle \quad (3.19)$$

with  $Q_1$  and  $Q_3$  being the first and the third quartile of the sample, respectively, and  $\Delta_Q$  interquartile range of the sample. The measured points are in Fig. 3.46.

We can see the strong negative bias present in “bumpy” measurements disappeared which supports our “angle enlargement” understanding of the phenomenon. The distribution of error  $\chi$  is plotted as histogram and Q-Q plot in Fig. 3.47. From the plots a slight negative skewness could be recognized, however, we propose that to be caused by too small sample size. None

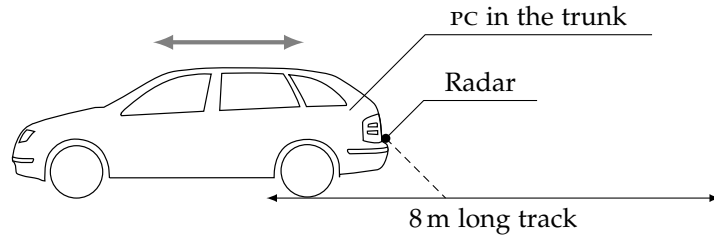


Fig. 3.45: The setup for indoor car measurement.

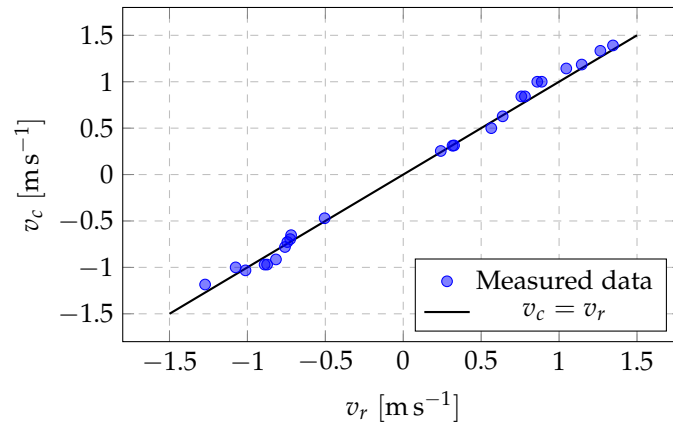
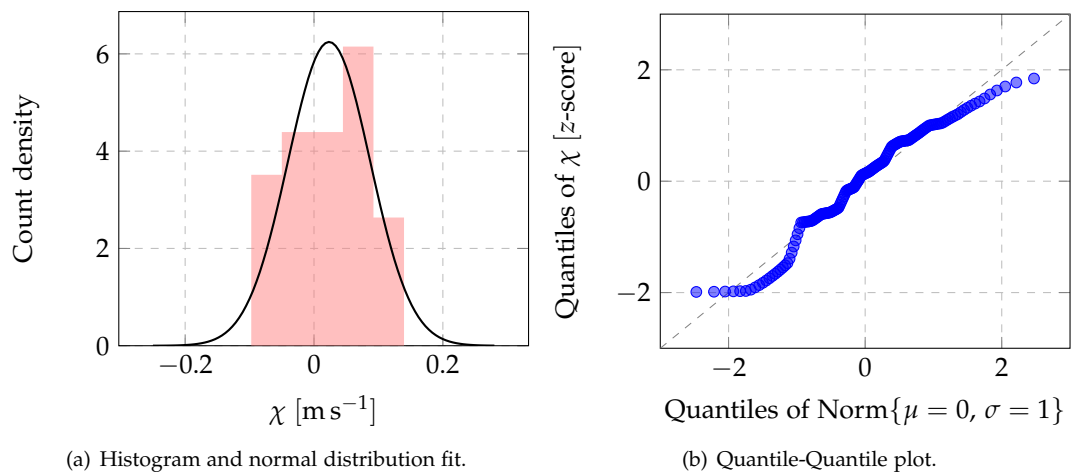


Fig. 3.46: Mean radar velocity  $v_r$  and calculated average velocity during a measurement  $v_c$ .

of Shapiro – Wilk and D’Agostino – Pearson tests rejected normality of the error with their respective results ( $W = 0.97$ ;  $p = 0.55$ ), and ( $K^2 = 0.91$ ;  $p = 0.64$ ) so we consider the error being gaussian. The LSQ fit of error results in  $\mu = 0.023 \text{ m s}^{-1}$ ,  $\sigma = 0.064 \text{ m s}^{-1}$  from which we can conclude again (as with the indoor measurements results) that the actual radar horizontal velocity results are *slightly underestimated*.



(a) Histogram and normal distribution fit.

(b) Quantile-Quantile plot.

Fig. 3.47: Distribution of error  $\chi = v_c - v_r$ .



**Outdoor car measurement.** Another approach to measurement of the *true* velocity has been used in measurement on a car outdoors. The setup has not been different from that in Fig. 3.45; the measurement itself, however, has been carried out differently.

Before a measurement started, a GPS tracking had been started. The built-in GPS receiver of *Xiaomi Mi A2 lite* smartphone have been used together with *Locus Map Pro* application [42] for tracking. In the application, one-second tracking interval (as the most dense of the possible ones) had been chosen. The track was being stored in a *gpx* [43] file. The surface was a dry asphalt road without significant damages. Some small holes or stones were occasionally present. From the point of view of GPS precision, the environment has been very kind; The sky was clear, and the space above the car has not been shielded by anything but some sparse shrubs and trees.

Once the GPS recording was running, a driver of the car began a drive and an operator of the script controlling radar started the measuring script. After a few minutes of driving in both the forward and the backward direction, the operator terminated the measuring script, and the track recording was stopped in the tracking application. The synchronization between starts and stops of the tracking application and the radar measuring script did not have to be ensured: Since in *gpx* files also time stamps are stored, and the measuring script stores them also, the synchronization could be done easily in postprocessing. The file format of the radar recording for the outdoor car measurement could be described as

- Three columns: `v_radar`, `time`, and `meas_number`.
- Delimiter is a single space.
- The first column is a float representing velocity in  $\text{m s}^{-1}$ , the second column is a time stamp formatted as `'%Y-%m-%d,%H:%M:%S'` [44], and the third column is an unsigned integer.

The postprocessing script takes an advantage of the `gpxpy` [45] python module for handling *gpx* files and works as follows:

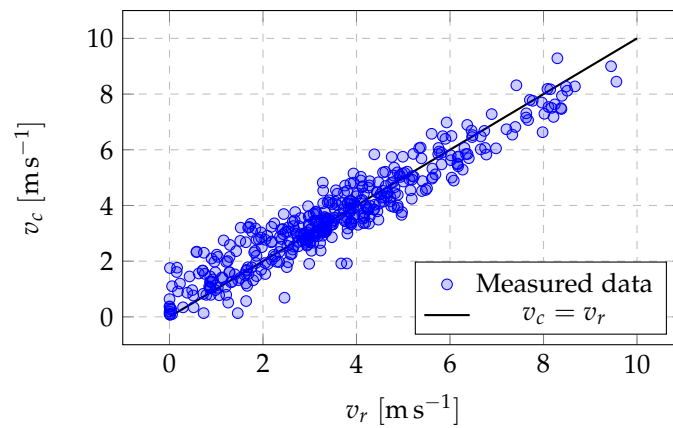
1. The *gpx* file from the measurement is parsed using `gpxpy`.
2. Using `gpxpy`, the velocity between each pair of the trackpoints is calculated, and the average time between the respective two trackpoints is assigned to the velocity value.
3. The result of the *gpx* processing is a `pandas.Series` object indexed by the time stamps, storing the calculated *true* velocity values.
4. The radar data are loaded to a `pandas.DataFrame` and the index is changed to `pandas.DatetimeIndex` according to the `time` column values.
5. The records of the radar data are grouped by index and the `v_radar` values are averaged within the groups. This causes the radar results are split into one-second intervals with `v_radar` being an average estimated velocity in them.
6. A new data frame `data` is created. Its index is a copy of the *gpx*-based data and its first column—`v_gps`—is a copy of the calculated true velocity values in times given by index values. The third column is labeled `v_radar` and is initialized by `np.nan`.
7. In a loop over the data index `i`, an index of the radar data is searched for a presence of a timestamp within time interval  $i \pm 5$ . If such index is found, the `v_gps` column on index `i` is updated with the respective `v_radar` value.
8. The entries of `data` which `v_radar` value is `np.nan` are dropped, which results in a final data frame with entries consisting of time-aligned (within the  $\pm 0.5\text{s}$  interval) values of estimated radar velocity and GPS-based velocity, considered as *true*.

The processing program is available as `cd/meas_scripts/meas_gps/process_gps.py`. A disadvantage of the approach is that the velocity from GPS measurements is not signed – therefore, we have absolutely neglected the sign and worked with an absolute value of the radar velocity

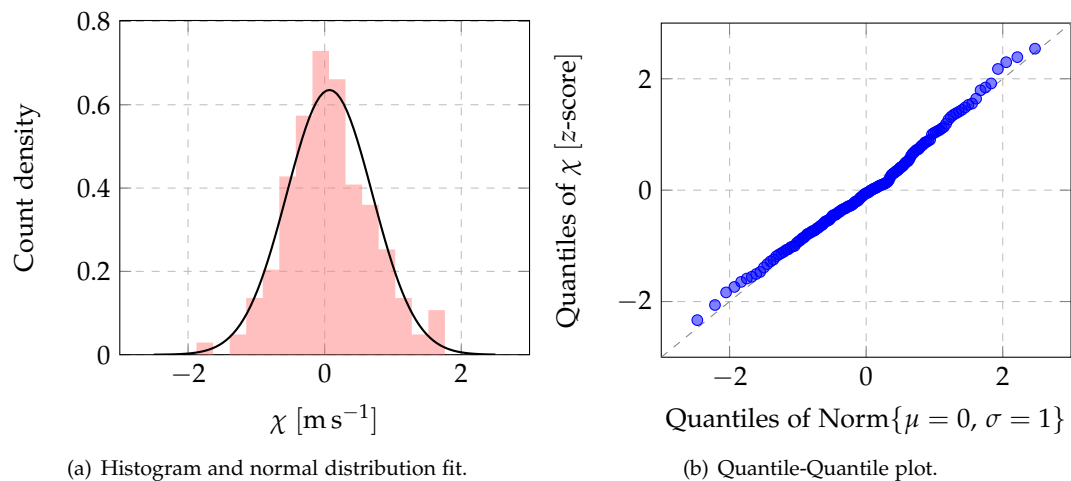
$v_r$  in data evaluation. The measured points are in Fig. 3.48, and the Q-Q plot and histogram of the error  $\chi = v_c - v_r$  are in Fig. 3.49(b) and Fig. 3.49(a).

D'Agostino – Pearson normality test *confirmed* normality of data with result ( $K^2 = 3.56$ ;  $p = 0.17$ ). Fit of the error by gaussian distribution resulted in  $\mu = 0.071 \text{ m s}^{-1}$ ,  $\sigma = 0.628 \text{ m s}^{-1}$ .

The observed standard deviation in this approach is much bigger than that of the indoor car measurement. By observing the values of  $\sigma$  of the fit by gaussian distribution for  $v_c$ -restricted data we found out that the value of  $\sigma$  does not change with  $v_c$  significantly. To conclude the measurement, it has to be pointed out that it has been carried out in conditions very close to those in which the radar speedometer could be employed. The precision of radar velocity estimation for velocities up to approximately  $10 \text{ m s}^{-1}$  could be given by a  $2\sigma$  value as  $\pm 1.256 \text{ m s}^{-1}$ . Nevertheless, the result assumes the GPS velocity is perfectly correct, which is in fact not likely to be true.



**Fig. 3.48:** Measured pairs of GPS velocity  $v_c$  and radar velocity estimate  $v_r$ .



(a) Histogram and normal distribution fit.

(b) Quantile-Quantile plot.

**Fig. 3.49:** Distribution of error  $\chi = v_c - v_r$  and its fit by normal distribution.

The outdoor trolley test revealed that the radar with the tested configuration cannot be used on some surfaces without additional calibration. All the other tests resulted in slight bias of the model from (3.16) – it is therefore appropriate to shift it. Since in any of the test we did not have an absolute velocity caliber, we decided to increase the quotient of the model by the lowest of

error means of the indoor trolley test and car tests, which is  $0.012 \text{ m s}^{-1}$ . The corrected model is

$$v_x = 1.176\mu_D - 17.676. \quad (3.20)$$

### 3.4.5 Stability of measurements, filtering results

On one hand, there is performance of the radar speedometer in terms of accuracy and precision, on the other hand, there is its *stability* in terms of not providing occasional “random” or inappropriate results. These wrong results could be caused by various phenomenons; namely unexpected objects on a road, or a packet slip in UART communication. In this subsection, we will analyse recordings of velocity evaluated using the scheme that have led to (3.20), and design a filter which will be applied at the very output of the processing chain.

### 3.4.6 Measuring velocity of a car

The first recording contains velocity data for a car driving on an asphalt surface. The surface has been the same as with *Outdoor car measurement* paragraph of Sec. 3.4.4. Asphalt cover has been in good conditions; some bumps were present though, as well as fallen branches and leaves once in a while. The car has traveled the total distance of approximately 2.9 km, most of that driving forward. Once, the car switched to reverse drive for a short time, and once the car turned back on the road, changing driving forward and backward in short intervals.

Data records from the experiment are vectors consisting of *timestamp* and *velocity* in  $\text{m s}^{-1}$ . They are gathered using the script in `cd/meas_scripts/stability_measurements/main.py`. The measured development of velocity in time could be seen in Fig. 3.51. Obviously, there are peaks present in the original data which are spurious and which have to be suppressed. It has been shown that *all* of the spurious peaks are only *one sample long*. They are probably caused by mistakenly read range – Doppler matrix message from UART. An easy and in the means of time consumption low-cost way of their removal is to watch a difference between the *present* sample and the *previous* sample, and if it exceeds the fixed threshold, assign the value of the previous sample to the present one. Such processing causes only one-sample delay. The threshold level has been set to exactly  $1 \text{ m s}^{-1}$ .

Another unwanted feature present in the original signal is a jitter – fast changes in  $v_x$  low in amplitude.<sup>xii</sup> Their amplitude does not change with  $v_x$ . To see whether they are spectrally flat or not, PSD (power spectral density) of the  $v_x$  signal has been estimated. For that, Welch’s method [46] with 128 samples per segment, 1024 samples DFT length (zero-fill), and linear detrending has been employed. To be competent to use the Welch’s method, we had to ensure that the signal is spaced evenly. Even though that is not absolutely true, the interval between samples as a random variable is normally-distributed, has mean of  $\mu_\Delta = 0.25 \text{ s}$  and standard deviation of  $\sigma_\Delta = 1.308 \text{ ms}$ . Its spread is that small that we can consider the sampling being even. The estimated PSD is in Fig. 3.50. The PSD revealed that the jitter is spectrally flat. Therefore, filtering in time-domain is appropriate to deal with it.

We have put two requirements on the jitter-removing digital filter:

- Suppress the jitter.
- Do not cause significant delay.

For such application, a moving average filter has been a clear choice. The only parameter to design has therefore been its length. The compromise between the two requirements mentioned above led to length of 5 samples. The numerator coefficients of the 5-sample moving average FIR filter’s impulse response are  $\frac{1}{5}(1 \ 1 \ 1 \ 1 \ 1)^T$ . The complete filtering chain (spikes removal, moving average) causes total delay of  $5 \cdot 0.25 \text{ s} = 1.25 \text{ s}$ , which we consider acceptable.

<sup>xii</sup>In the measured sample, their amplitude does not exceed  $0.6 \text{ m s}^{-1}$ .

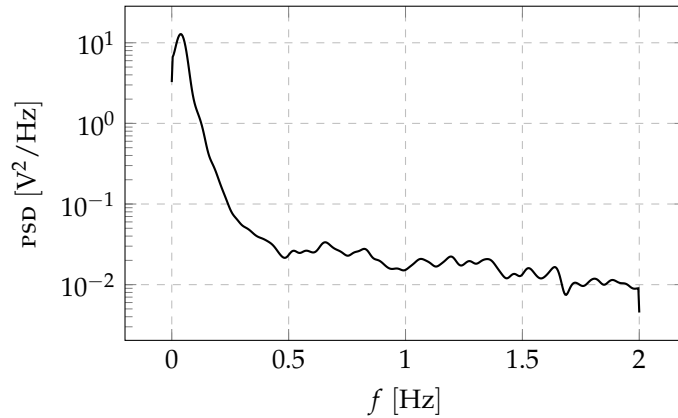
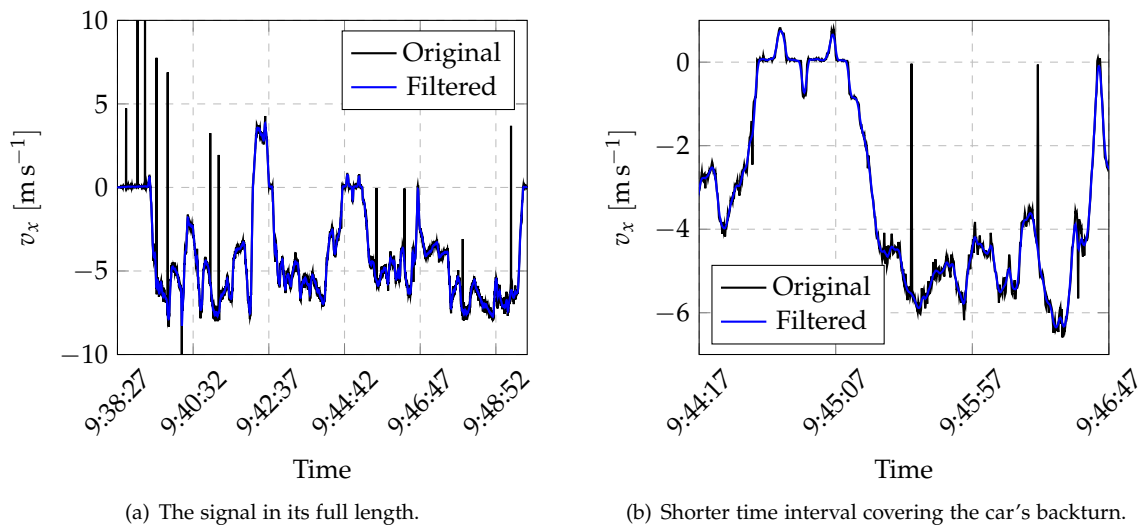


Fig. 3.50: Velocity data PSD estimate using Welch's method.



(a) The signal in its full length.

(b) Shorter time interval covering the car's backturn.

Fig. 3.51: Measured horizontal velocity of the car with and without filtering applied.

The last spurious feature which could be seen in the data is a bias of approximately  $0.15 \text{ m s}^{-1}$ . In this case, it is a bias in terms of *accuracy*, unlike with the bias corrected in (3.20) which has been caused by poor *precision*. In order to suppress it, we measured velocity in a static case, when no motion was present.

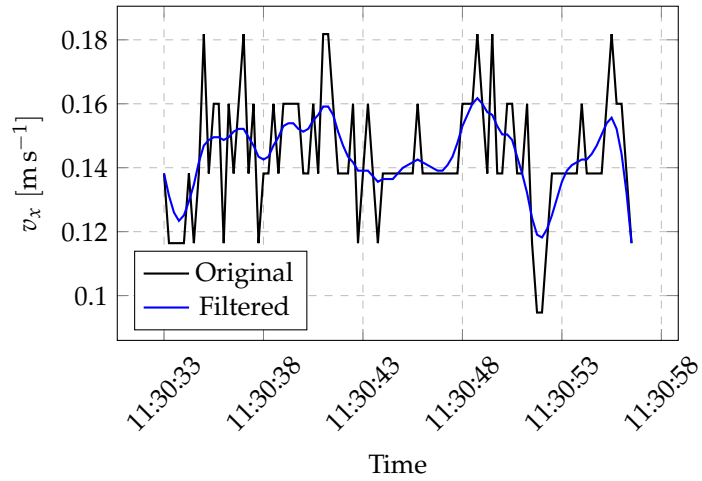
### 3.4.7 Measurement of bias

The measurements for correcting the bias from the previous subsection were realized using the indoor trolley setup (see Sec. 3.4.4). The trolley was standing on the fixed position for approximately 25 s, and during that epoch horizontal velocity was estimated. The results are plotted in Fig. 3.52. It is obvious that a bias of approximately  $0.15 \text{ m s}^{-1}$  is present.

To manage for its correction, we subtracted the bias term  $0.15 \text{ m s}^{-1}$  from the model (3.20):

$$v_x = 1.176\mu_D - 17.838. \quad (3.21)$$

Moreover, to ensure that the bias could be easily removed even if it—for any reason—got value



**Fig. 3.52:** Static velocity measurement from indoor trolley setup. Both unfiltered, and MA(5)-filtered results.

other than  $0.15 \text{ m s}^{-1}$ , we have implemented a bias calibration in the *Velocity viewer* GUI described in Sec. 4.2.2.

## Software implementation

A software that provides the whole second-order<sup>i</sup> signal processing and data visualisation has been written. The software is written in *Python 3* extended by the *NumPy* library and some standardly used libraries simplifying the tasks like serial communication or plotting (*Pyserial*, *PyQtGraph*, etc.). It is divided into modules those we can basically subdivide into the *backend* modules, serving for gathering the input data and computation of the results, and *frontend* modules which visualize the results, or store them for later use.

### 4.1 Backend modules

The backend modules of the program cover these three tasks:

1. Representation of the binary data from the EVM's UART output with readable structures (data parsing)
2. Calculation of the velocity values from the parsed data
3. Communication with a remote PC where the results are displayed.

Their implementation will be explained in the following subsections.

#### 4.1.1 Data parser

The function of the data parser is arranged by the `Packet_handler` module. The `Packet_handler` requires standard modules `numpy`, `time`, `struct`, `serial`, `os`, `ast`, and `sys`. It defines the `Packet_handler` class that contains these entities:

**The constructor** requiring one mandatory and one optional user arguments: The mandatory argument `config_file` (string) is a full path (including the suffix) to the configuration file by which the EVM has been configured. The class needs it to derive the parameters of the chirp train which influence a format of the UART data (number of range bins, number of chirps per frame, sampling frequency, etc.).

The constructor opens the configuration file and from the relevant lines of that it extracts/calculates these numbers:

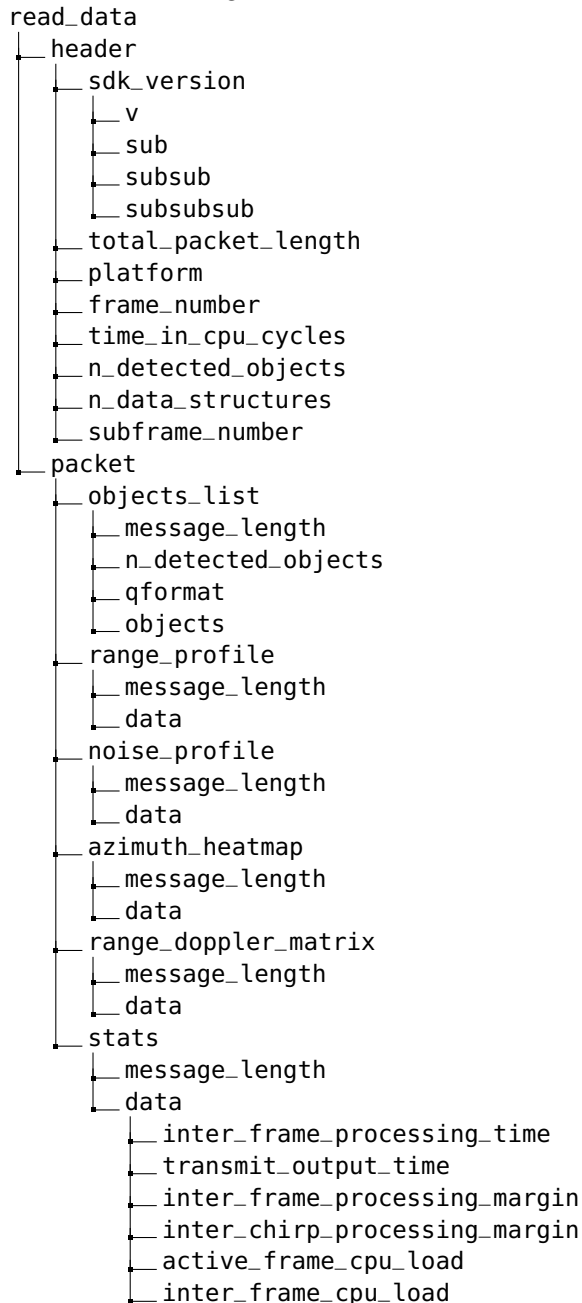
- Number of RX and TX antennas ( $N_{RX}$  and  $N_{TX}$ , respectively) from the `channelCfg` line
- Starting frequency of the chirp  $f_1$ , idle time  $T_i$ , ramp time  $T_r$ , frequency slope  $S$ , number of range bins  $N_{FFT}$ , and the sampling frequency  $f_p$  from the `profileCfg` line
- Number of subframes per frame  $N_{SF/F}$ , and number of chirps in one subframe  $N_{c/sf}$  from the `frameCfg` line.

<sup>i</sup>The processing of the data structures outputting the TI AWR1642BOOST.

From these numbers and the physical constants, some other parameters are calculated, namely:

- Number of chirps in a frame as  $N_{C/F} = N_{SF/F} N_{C/SF}$
- Number of Doppler bins as  $N_{C/F}/N_{TX}$  [47]
- Number of virtual antennas as  $N_{TX} N_{RX}$
- Range constant  $C_r$  and Doppler constant  $C_d$  using (3.1).

The `get_new_packet` function with no user arguments returns a dictionary filled by the values of one UART data packet as described in subsection 3.1.3. The returned dictionary has a fixed form matching the definition, with the following labels:



Firstly, the function opens the serial port `/dev/ttyACM1`, having baudrate set to 921 600 Bd. It defines an 8 bytes long buffer `MAGIC_BUFFER`, and then in an infinite loop

1. Reads 1 B from the serial port and interprets it as an unsigned character (unsigned one-byte integer)
2. Shifts the MAGIC\_BUFFER one position to the left
3. Puts the value to the very end of the MAGIC\_BUFFER.
4. Checks if the content of the MAGIC\_BUFFER equals the constant MAGIC\_WORD. If it does, the function breaks the infinite loop.

Next, the particular cells of the read\_data dictionary are filled accordingly with the Fig. 3.14 and the structure definitions from Appendix A.

#### 4.1.2 Range – Doppler matrix processor

The employed velocity evaluation algorithm (Mean of Doppler bins argmax, Sec. 3.2.1) is based on a range – Doppler matrix (RDM) data. After obtaining the packet by calling the Packet\_handler.get\_new\_packet function, the following operations are made on the RDM data in order to transform them in accordance with Fig. 3.17:

**Transposing the original data.** The RDM data are transmitted bin-by-bin in the range dimension: First, all the Doppler bins for the first range bin are transmitted, then the structure continues with the second range bin, etc. The received matrix therefore has a form displayed in Fig. 4.1.

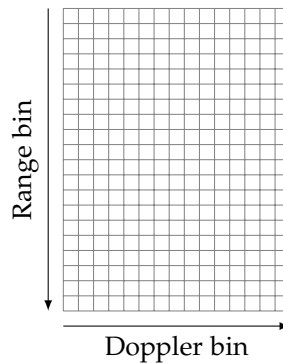


Fig. 4.1: The form of range – Doppler matrix as received from UART.

**Rearranging the rows of the transpose.** Since we would like to have the zero Doppler bin in the middle, the Doppler bins associated with positive velocity in the first half (velocity raising from the middle to the top), and the Doppler bins with negative velocity in the lower half (velocity decreasing from the middle to the bottom), we need to rearrange the rows of the new matrix. Firstly, the matrix is “cut” in the middle and the halves are swapped. In the next step, the new structure is flipped upside down. The steps are depicted in Fig. 4.2.

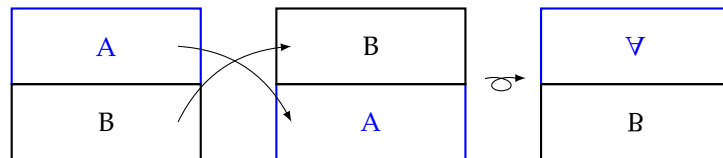


Fig. 4.2: Rearrangement of the rows of the transposed matrix.

From now on, we will refer to this transformed matrix simply as range – Doppler matrix.



### 4.1.3 Bluetooth communication module

The *Bluetooth* communication is supervised by the `Bluetooth` module based on the *Python's* `socket` module. Besides that, it imports the *NumPy* module. It defines the `Bluetooth` class, containing the following functions:

The **constructor** is empty.

The **connect method** takes two user arguments:

1. `address` (string) – The `MAC` address of the *Bluetooth* adapter of the remote PC<sup>ii</sup>
2. `port` (integer) – The port number to connect to.

It uses the `socket` module to make a new socket instance and to connect it to the socket defined by the pair (`address`, `port`).

The **close method** Takes the open socket instance as a user argument and closes it.

The **send function** requires 2 user arguments. One of them is an open socket instance, the second is called `content` and it is a *bytes array* to be sent. Its return value is inherited from the `socket` class and is *True* for success and *False* for an error.

The **receive function** is intended to be ran on the remote PC. It takes these user arguments:

1. A physical address of a *Bluetooth* adapter (string). The valid address format consist of 6 hexadecimal pairs delimited by colon: `xx:xx:xx:xx:xx:xx`.
2. A port number (integer).
3. Number of lines (words) to be received (integer). In terms of velocity values, the number of velocity samples to receive.
4. A maximum size of each line in bytes (integer).
5. Backlog (integer). In our case of only one client, it should be set to 1 which is also its default value.

The return value is a *NumPy* array of 32bit integers containing the received lines. According to the data type of the array, it is clear that only values interpretable as 32 bit integers could be a content of the transmitted lines.

The **string\_to\_bytes auxiliary function** encodes the string given as an argument as bytes using the *utf-8* standard. It returns the encoded byte array.

### 4.1.4 Utility module

There is one extra module that has been created. It contains auxiliary methods and exceptions used in the modules directly connected with the velocity evaluation. The module is called `Utils` and imports community modules *NumPy*, `matplotlib.pyplot`, `os`, `sys`, and `time`. The module defines an `Utils` class with an *empty* constructor. Further the class provides:

**WrongGuiMonitorConfigException** which is an alias for the general `Exception` class. Its purpose is to be raised whenever the *EVM* configuration file given does not match with the structure of the demo firmware configuration files as described in section 3.1.2.

**FileException** which is an another alias for the `Exception` class. It is raised if the file on the given path does not exist.

The **recursive\_keys** generator takes a dictionary as its only user argument and yields recursively the given dictionary's keys. It is useful when testing for a presence or absence of a key in a nested dictionary – just like the one returned from the `get_new_packet` function of the `Packet_handler` class.

---

<sup>ii</sup>Generally speaking, it could be an address of *any Bluetooth* device which the PC running the `Bluetooth` module could connect to. Nevertheless, here we address the particular case of a remote PC since that is the setup we are aiming to.

The **peak\_function** is a wrapper for the *NumPy* `argmax` function. It is defined for a case when another definition of a *peak* is needed than the one provided by the *NumPy* `argmax` function.

Functions **range\_index\_to\_meters** and **doppler\_index\_to\_mps** which convert the index (integer or float, the first user argument) to value in SI units. Both of the functions have the same argument syntax:

- The second user argument is a path (string) to the configuration file according to which parameters the index should be converted.
- The third (optional) argument, `reverse` (boolean), allows to switch the direction by its `True` value and make the functions return indices when the SI values are put on the place of the first argument.

## 4.2 Front end

During the testing phase, there has been a need for visualizing UART data. Although there is a visualization tool provided by the manufacturer [48], it does not offer the full functionality we have needed. The tool could provide a connection to the configured device, receive UART data and display plots of transmitted data structures. However, there is basically *any* option for plotting the quantities we calculate on the top of the basic plots, or even modify the plotted data. We decided to implement a complete visualization tool providing all the requested functionality.

Another important part of the front end part is a GUI that allows an unexperienced user use the whole speedometer as a blackbox, filling only some of the most necessary fields and clicking some self-explanatory buttons. He should be able to see the velocity results, and optionally store the measured data for later use.

### 4.2.1 The Plotter module

The mother module of the customizable data visualization part is called `Plotter`. The module imports some standard *Python* modules (`time`, *NumPy*, `pandas`), two plotting modules (`matplotlib.pyplot` (plus `matplotlib.animation`) and `pyqtgraph`), and many of our own submodules, each one handling one particular plot type. Some of the less demanding plots (with respect to the amount of data and the plot's refresh rate) are implemented in the more user-friendly *Matplotlib's* `pyplot`, the more complex ones use the significantly faster `pyqtgraph` module. These `pyqtgraph`-based plots are the ones implemented as standalone modules.

The `Plotter` class consists of the following functions:

**The constructor** that instantiates member objects `Plotter.fig` and `Plotter.ax` for plotting the `matplotlib.pyplot`-based plots, a `Packet_handler` object as a source of data, and a `Utils` object for performing some of the auxiliary operations.

**scatter\_xyz function.** The function does not request any user arguments because it operates on the member `fig` and `ax` objects. It does neither return anything. It defines an `animate_scatter_xyz` function which is an animation callback to be fed to the `matplotlib.animation.FuncAnimation` method. The callback itself

- Calls the `Packet_handler.get_new_packet` function and stores its return value.
- Extracts the *objects list* message out of it.
- Plots a scatter plot described by Fig. 4.3 to the `(fig, ax)`-pair.

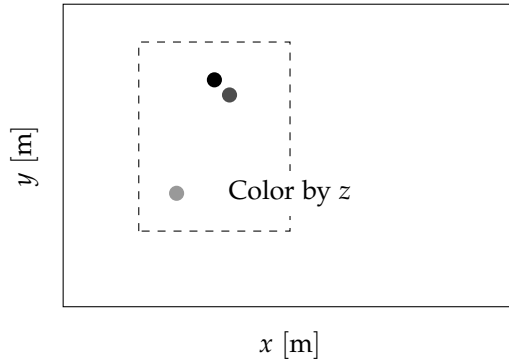


Fig. 4.3: Format of the scatter plot of detected objects.

**range\_profile** function. The function has no mandatory user arguments but could be given 3 optional keyword arguments: `noise_profile_on`, `detected_objects_on`, and `grid_on`. It operates over *detected objects*, *range profile*, and *noise profile* messages and so the `guiMonitor` configuration has to reckon with it.

After performing tests for presence of all the needed data in the UART packet (raising `Utils.WrongGuiMonitorConfigException` if it detects a problem there), the function operates on the member `fig` and `ax` objects via a `matplotlib.animation` callback function. The callback does the following:

- Calls the `Packet_handler.get_new_packet` function and stores its return value.
- Generates a range axis vector as  $(0, \frac{f_p c_0}{2S}, num = L)$  where  $L$  is a value of 'message\_length' field of the 'range\_profile' message divided by 2 (The length is in bytes and the range profile values are expressed as 2 B integers).
- Plots the range profile points in an appropriate range axis to the (fig, ax)-pair.
- If the `detected_objects_on` argument is set to `True` (default is `False`), also the noise profile data (the row of a range - Doppler matrix with the highest Doppler index) is plotted.
- If the `detected_objects_on` is set to `True` (default is `False`), range profile peaks corresponding to the detected objects are marked by an arrow.
- If the `grid_on` argument is set to `True` (default is `False`), the final plot contains a grid.

A sketch of the plot is in Fig. 4.4.

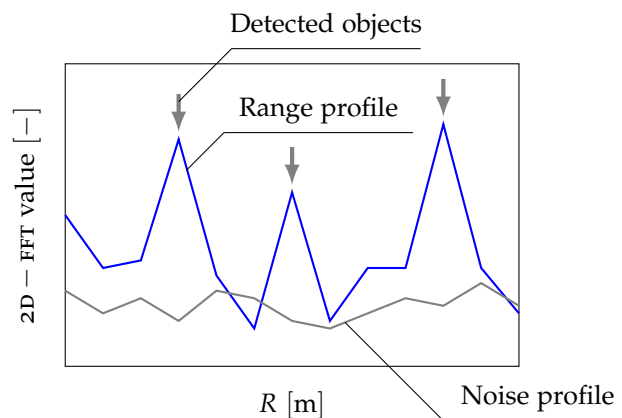


Fig. 4.4: Format of the range/noise profile plot.

**range\_doppler\_heatmap function.** This function only creates an instance of the `Range_doppler_heatmap` class and calls its `run` method. This kind of plot does not use the member `matplotlib`-based `fig` and `ax` objects. The data to be visualized (order of  $1 \cdot 10^3$  values) and their refresh rate (approximately twice a second) is too heavy for `matplotlib.animation` and we need to employ the faster `pyqtgraph` module. The rest of this description addresses the `Range_doppler_heatmap` class itself.

The constructor has 9 optional user arguments:

1. `figsize`. A tuple (width, height) of the displayed figure in pixels. Default is (1280, 1024).
2. `minmax`. A tuple (minimum, maximum) with limits of the lookup table for assigning colors to the cells of the RDM. Default is ( $1.5 \cdot 10^4$ ,  $3.5 \cdot 10^4$ ).
- 3.–4. `n_xticks`, `n_yticks`. Number of horizontal and vertical axis ticks (granularity of the axis division). The ticks are separated uniformly.
5. `detected_objects_on`. When set to `True`, the cells of the RDM which passed the `CFAR` and peak grouping algorithms are marked by diamonds. Default is `False`.
6. `maxima_on`. A flag for rendering a red line on a Doppler position (through all range bins) equal to the mean of Doppler `argmax` across the range bins. See Sec. 3.2.1 for details.
7. `centroid_on`. A flag for displaying a green line connecting centroids in Doppler dimension across all the range bins. See Sec. 3.2.2 for details.
- 8.–9. `color` and `grid`. Both are boolean flags and control colormap and grid presence, respectively. If `color` is `True`, the RDM uses the `jet` colormap. Otherwise, the `gray` colormap is used. Defaults are `color = False`, `grid = True`.

By the constructor, a `Qt` application layout is set using `pyqtgraph` objects and functions focused specially on plotting. A `pyqtgraph.PlotItem` is created for possible detected objects, lines regarding horizontal velocity evaluation algorithms, and axis ticks. In its axis, a `pyqtgraph.ImageItem` is instantiated for displaying a RDM.

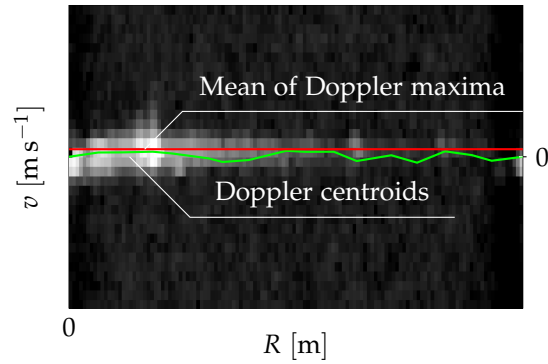
A `Packet_handler` instance calls its `get_new_packet` function and the plot is initialized: The RDM structure is rearranged using the scheme from Fig. 4.2 and displayed as an image. On the top of it, the diamonds marking *detected points* and the horizontal lines are plotted if it is requested by the respective argument values. A `Qt` timer is created, its timeout signal is connected to the `Range_doppler_heatmap.update_image` and it is started with timeout set to 100 ms.

The `update_image` member function (no user arguments) retrieves a new `UART` packet using the `Packet_handler.get_new_packet` method, rearranges the RDM, fills the `ImageItem` by the new rearranged RDM, and plots the other requested features on its top (if requested).

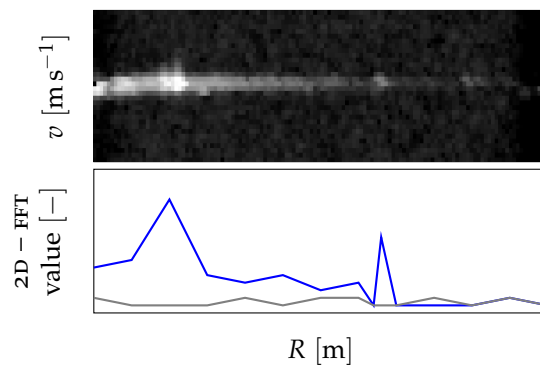
The last method of the class—`run`—only calls the `Qt` application's `exec_` method. The range – Doppler heatmap plot with its optional features is drawn in Fig. 4.5.

**range\_doppler\_heatmap\_and\_range\_profile function.** Since we have found it useful to see a RDM and range profile stacked vertically, we have implemented a `pyqtgraph`-based plotting module `Range_doppler_heatmap_and_range_profile`, which is handled by this function from within `Plotter` class. The function creates an instance of the class and calls its `run` function. The function—as well as a constructor of the class—has four optional keyword user arguments: `figsize`, `minmax`, `n_xticks`, and `n_yticks`. Their meanings and default values are the same as with the `Range_doppler_heatmap` constructor described before. The resultant plot content is drawn in Fig. 4.6.

**doppler\_profile function.** Doppler profile is an equivalent of range profile in Doppler dimension. Range profile itself is transmitted as a separate message within `UART` data. Doppler profile is *not* and is only acquired as a cut of a range – Doppler matrix. It is feasible to handle 2D data of Doppler profile using `matplotlib.pyplot/animation` `fig` and `ax` pair so no special class for this kind of plot is needed. One mandatory argument and two optional arguments are accepted by the function: The mandatory one is



**Fig. 4.5:** An appearance of a plot generated by `Plotter.range_doppler_matrix` call.



**Fig. 4.6:** The schematic of a `range_doppler_heatmap_and_range_profile` function.

`range_index` as an integer index value impacting *where* the RDM is cut. The optional arguments are flags `usemeters` (default is `False`) and `grid_on` (default is `False`). While the former resolves between use of range/Doppler indices or SI units, the latter simply turns on a grid in a plot.

The function uses the `Plotter`'s member `Packet_handler` instance to get a new packet. It tests for presence of range – Doppler matrix data. If the test passes successfully, the function initializes the canvas (`ax`) with labels, limits, grid (optionally), and a title. Then, similarly as with other `matplotlib`-based plots described before, a plot refresh callback is called repeatedly. The callback is called `animate_doppler_profile` and its workflow follows the following steps:

1. Getting new packet.
2. Rearranging the RDM data (see Fig. 4.2).
- (3). (*Only in the first call*) Setting axis limits with respect to the particular cut of rearranged RDM.
4. Plotting the cut using the `Plotter`'s (`fig, ax`)-pair.

The result has a form depicted in Fig. 4.7.

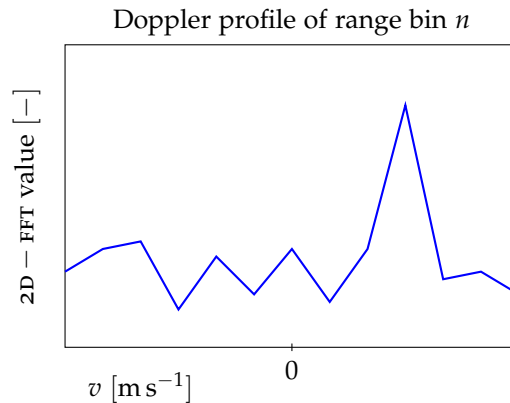


Fig. 4.7: A form of the `Plotter.doppler_profile` function result.

**velocity\_progressive\_plot function.** Eventually, this could be called the most important function of the whole `Plotter` module because it has the knowledge of the sections addressing velocity estimation on its background. Less pompously, it implements the algorithm of velocity evaluation using (3.21), and displays the results. It instantiates the `Velocity_progressive_plot` class and has the same user arguments:

1. `config_file`. A path to the file used for the radar configuration.
2. `csv_filename`. This is a filename of a file to which the results will be stored. The results are in a form of a space-delimited file with `time` and `velocity` columns. The time column is in `'%Y-%m-%d,%H:%M:%S.%f'` format, taking only the first three digits of the seconds' fractional part into account. The `velocity` column contains values of measured horizontal velocity in  $\text{m s}^{-1}$ . The default value is `None` which means no file is created and no results are saved.
3. `moving_average` (Integer). Number of subsequent cells to calculate the moving average from. The reasoning for filtering the results is given in 3.4.5. Default is 0 which has a meaning of "do not apply any filtering". For any nonzero value of the `moving_average` parameter, the module automatically filters possible one-sample "spikes," as those in Fig. 3.51(a) by comparing the values of two subsequent velocity samples and thresholding the value by a constant  $1 \text{ m s}^{-1}$ .

The resulting plot is a red line updating with every newly-arrived packet. When the line reaches the end of the axis (100 samples), the line is cleared and the plotting continues in clear axis.

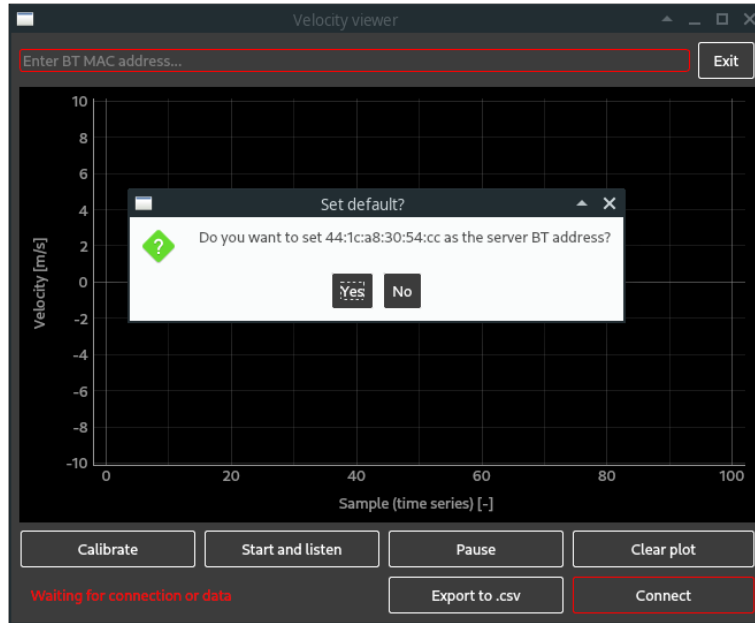
This kind of plot is employed in the *Velocity viewer* GUI (Sec. 4.2.2).

#### 4.2.2 Velocity viewer GUI

The GUI we have developed has a goal to allow an user to

- connect to the *Raspberryπ* via *SSH* and *Bluetooth* easily
- view the course of velocity in a simple  $t - v_x$  plot
- save the measured velocity to a *csv* file.

All of that on a few clicks. The GUI is designed using `PyQt5` library and takes advantage of `pyqtgraph` plotting module. The GUI is called *Velocity viewer*. After executing the application, the main application window is open and immediately a "Set default?" dialog pops up, as could be seen in Fig. 4.8. We used the dialog during various measurements employing the *Velocity viewer* not to have to fill the *Bluetooth* address of the server every time the application has been executed.



**Fig. 4.8:** The initial screen of *Velocity viewer* application.

By clicking *Yes*, the default *Bluetooth* address `44:1C:A8:30:54:CC` is filled in the *Enter BT MAC address* line edit. Otherwise, an user has to fill another *Bluetooth* address to that line edit. The line edit accepts any hexadecimal twelve-tuple (possibly with colons delimiting the hex pairs). It is case-insensitive. Once a valid *Bluetooth* address is filled in, the line edit and the *Connect* button turn green. The latter also becomes *enabled* by it and an user can click on it (pressing return after filling the *Bluetooth* address in the line edit works as well). Clicking the *Connect* button opens another “*Set default?*” dialog that in this case targets to *SSH* connection parameters.

Establishing a *SSH* connection with the *Raspberry $\pi$*  serves for running the velocity evaluation program and computed velocity transmission once the radar *EVM* is configured. It is required to establish the connection only for a short time to allow some commands be executed – then, the connection is not used anymore and neither the *Raspberry $\pi$*  nor the host *PC* need to be connected to a common network then. The reason why the velocity evaluating/transmitting program is not run on a *Raspberry $\pi$* 's startup is that the *Raspberry $\pi$*  has to *know* the host's *Bluetooth* address before the transmission starts. Using *SSH* is a reliable way how to transmit all the needed information to the *Raspberry $\pi$* . There are two things to remember when using the system in a new *Wi-Fi* network: Firstly, the new network's name and password has to be saved to the *Raspberry $\pi$* 's `/etc/wpa-supPLICANT/wpa-supPLICANT.conf` file. Secondly, the *IP* address of the *Raspberry $\pi$*  in the new network has to be found out<sup>iii</sup>. The *Raspberry $\pi$*  authentication strings are

**Username:** pi  
**Password:** proactive impeach

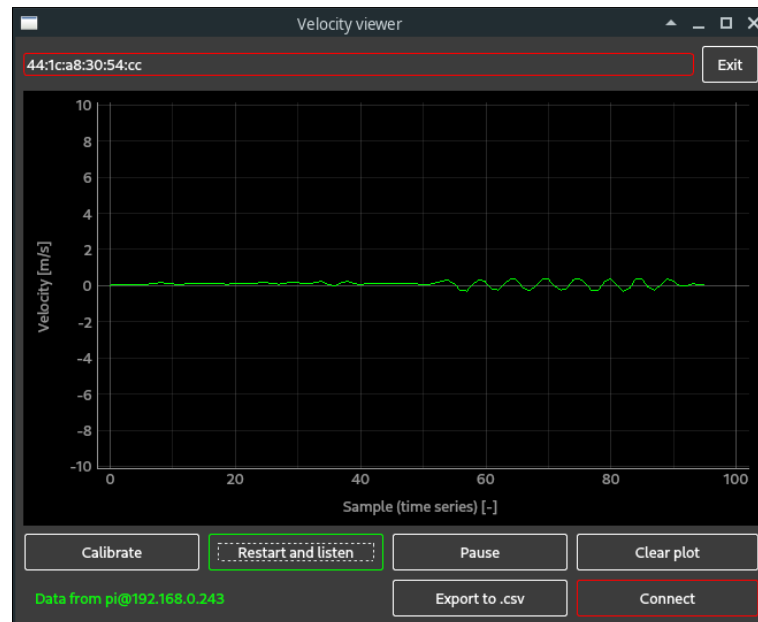
by default. If both the *Bluetooth* and the *SSH* connections are successfully established, the textbox at the lower left corner of the application changes from *Waiting for connection or data* to *Data from pi@<ip-address>* and the velocity samples are plotted in the axis. A screenshot of this situation is in Fig. 4.9. In addition to this very basic functionality the application provides some utilities that are turned on and off by clicking the corresponding button:

<sup>iii</sup>Probably the simplest way is running `arp -a` in the host *PC* shell.

**Pause and Clear plot.** The velocity plotting could be paused and resumed, or cleared.

**Bias calibration.** During the calibration, the mean of velocity samples is computed. When the calibration stops, the computed mean is subtracted from each of the following samples.

**Export to csv.** The measured data vectors (timestamp, velocity)<sup>T</sup> could be saved to a csv file. The filename is automatically set up based on actual date and time and is revealed to an user by an information box.



**Fig. 4.9:** The screen of *Velocity viewer* application when all connections are successfully established.

### 4.3 Documentation

The package of software tools for packet handling, visualization, etc. has been documented using the `pdoc3` tool for automatic generation of documentation from `docstrings`<sup>iv</sup>. The generated documentation is in `html` format. It is cross-referenced and could be easily opened in a web browser. The documentation is attached as `/documentation_html/` on the CD.

<sup>iv</sup>*Docstring* is a part of code similar to comment (in terms of it is ignored when a program is compiled/interpreted on its own) that has a special syntax and serves as source code for documentation generation tools.



## Conclusion

In this thesis, the basic principles of FMCW radars were reviewed, with a particular focus on the aspects influencing applicability of this kind of radars to measurement of horizontal velocity of an object travelling on a flat ground. To extend the theoretical part, we have derived an equation of an area from which the substantial part of a radar beam's energy reflects. We have used the equation later to design a proper placement of the radar on a moving object.

After exploring the general principles, we have moved our focus to the TI AWR1642BOOST radar evaluation module. The module's hardware and its software capabilities have been overviewed, and the particular firmware features have been chosen to take a part in the final radar software. Algorithms for estimating horizontal velocity from the data provided by the module have been implemented and tested, which eventually resulted in selecting the *Mean of Doppler bins argmax* algorithm as the one to be employed. Considerable effort has been put on design of the best-performing radar signal (chirp train). An impact of various chirp train parameters on quality of the resulting signal. Ratio between the peak and average power of a range – Doppler matrix cut at a particular range bin has been modeled as a function of sampling frequency and slope of a linear chirp. Empirically, it has been found out which of the possible chirp train configuration is the best for our application.

A compact radar speedometer employing the TI AWR1642BOOST module and a *Raspberryπ* computer has been designated and constructed with emphasis put on its independence (in terms of wired connections) on non-mobile devices such as laboratory power source, or a laptop. The velocity output could be sent to a remote device using the *Bluetooth* technology.

The relationship between the mean of Doppler bins *argmax* of a range – Doppler matrix and its horizontal velocity content has been appointed based on several measurements in various setups. After deriving the first form of the formula, another measurements were conducted to fine-tune precision and accuracy of the velocity estimate.

The device is designated for velocity band from  $-9.76 \text{ m s}^{-1}$  to  $9.76 \text{ m s}^{-1}$ . Its precision based on GPS reference has the  $2\sigma$  value of  $1.256 \text{ m s}^{-1}$ , and that based on measurements of time and distance has the  $2\sigma$  value of  $0.128 \text{ m s}^{-1}$ . Even though the velocity measurements using the device extended by a five-sample moving average filter exhibit very good stability, since we have not disposed with any *absolute* velocity caliber, quantification of precision could not have been done other way than using the other velocity-evaluation methods as a reference. Because horizontal velocity has a linear dependence on the statistic that the evaluation is based on (mean of Doppler bins *argmax* in a range – Doppler matrix), the velocity estimate after subtraction of a velocity estimate in static state, there should not be a problem with its accuracy.

To allow for using the device for real-case velocity evaluation, a variety of software tools for the radar kit's data handling and visualization have been written in *Python3*. The most important software tool for real deployment is a GUI allowing an user to setup the measurements on a few clicks and visualize or store the results.

The future work on this topic surely has to address quantification of the device's precision using some absolute velocity standard. Another improvements could be done for example on the design of the mounting box, to make the whole device more mechanically robust.

## UART data structure

Number	Name	Data type	Length	Note
1	Magic word	uint8	8	Fixed value [2,1,4,3,6,5,8,7]
2	Subsubsubversion	uint8	1	SDK version equals v . sub . subsub . subsubsub. In our case 2.0.0.4
3	Subsubversion	uint8	1	
4	Subversion	uint8	1	
5	Version	uint8	1	
6	Total packet length	uint32	1	Includes the header
7	Platform	uint32	1	Represented hexadecimally equals A1642
8	Frame number	uint32	1	
9	Time in CPU cycles	uint32	1	Number of DSP cycles at the time of the message creation
10	Number of de- tected objects	uint32	1	
11	Number of data structures	uint32	1	
12	Subframe number	uint32		

**Tab. A.1:** The structure of the packet header.

Number	Name	Data type	Length	Note
1	Tag	uint32	1	1 for the list of detected objects, 2 for range profile, 3 for noise profile, 4 for range – azimuth heatmap base, 5 for range – Doppler heatmap, 6 for performance info
2	Message length	uint32	1	In bytes

**Tab. A.2:** The structure of the message descriptor.

Number	Name	Data type	Length	Note
1	Number of detected objects	uint16	1	Has to equal the value from the header
2	Q format	uint16	1	$2^{(Q \text{ format})}$ is the divisor of the integer representation of $x$ , $y$ , and $z$ coordinates.
3	Objects data	Object structure (see Tab. A.4)	Number of detected objects	

**Tab. A.3:** The structure of the *detected objects* message (Tag = 1).

Number	Name	Data type	Length	Note
1	Range index	uint16	1	
2	Doppler index	int16	1	
3	FFT Peak value	uint16	1	
4	x	int16	1	Has to be divided by $2^{(Q \text{ format})}$
5	y	int16	1	Has to be divided by $2^{(Q \text{ format})}$
6	z	int16	1	Has to be divided by $2^{(Q \text{ format})}$

**Tab. A.4:** The structure of one detected object.

Number	Name	Data type	Length	Note
1	FFT sample	uint16	Number of FFT bins	The number of FFT bins is given by the configuration file. Alternatively, it could be calculated from the message length as its half (size of uint16 is 2 bytes)

**Tab. A.5:** The structure of the range profile message.

Number	Name	Data type	Length	Note
1	FFT sample	uint16	Number of FFT bins	The number of FFT bins is given by the configuration file. Alternatively, it could be calculated from the message length as its half (size of uint16 is 2 bytes)

**Tab. A.6:** The structure of the noise profile message.

Number	Name	Data type	Length	Note
1	Detection slice at zero Doppler	cube (int16 imaginary part, int16 real part)	complex32 (int16 imaginary part, int16 real part)	Number of FFT bins multiplied by the number of virtual antennas for the 0th range bin first, then that for the second range bin, etc.

**Tab. A.7:** The structure of the azimuth heatmap source samples message.

Number	Name	Data type	Length	Note
1	Sample	uint16	The number of range bins multiplied by the number of Doppler bins	All Doppler bins for the 0th range bin first, then for the 1st range bin, etc.

**Tab. A.8:** The structure of the range – Doppler heatmap message.

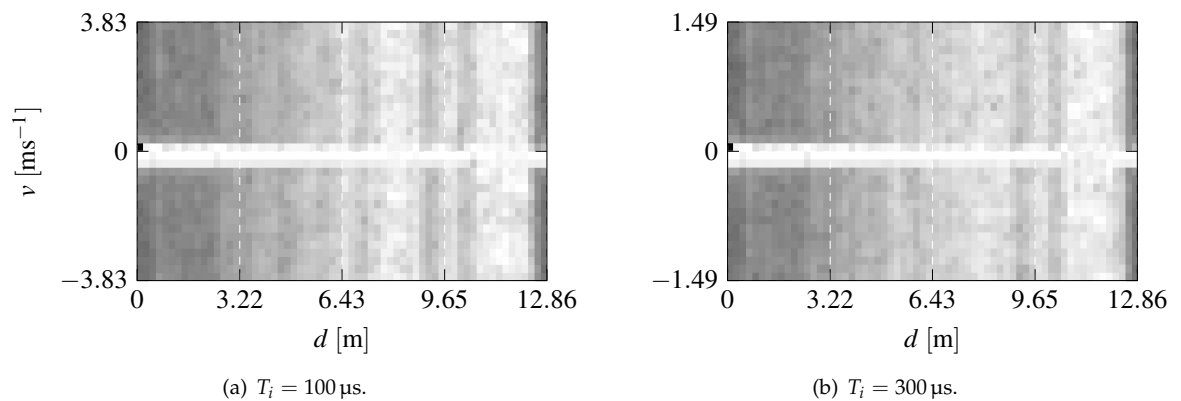
Number	Name	Data type	Length	Note
1	Interframe processing time	uint32	1	
2	Transmit output time	uint32	1	
3	Interframe processing margin	uint32	1	
4	Interchirp processing margin	uint32	1	
5	Active frame load	uint32	1	
6	Interframe load	uint32	1	

**Tab. A.9:** The structure of the processing performance (stats) message.

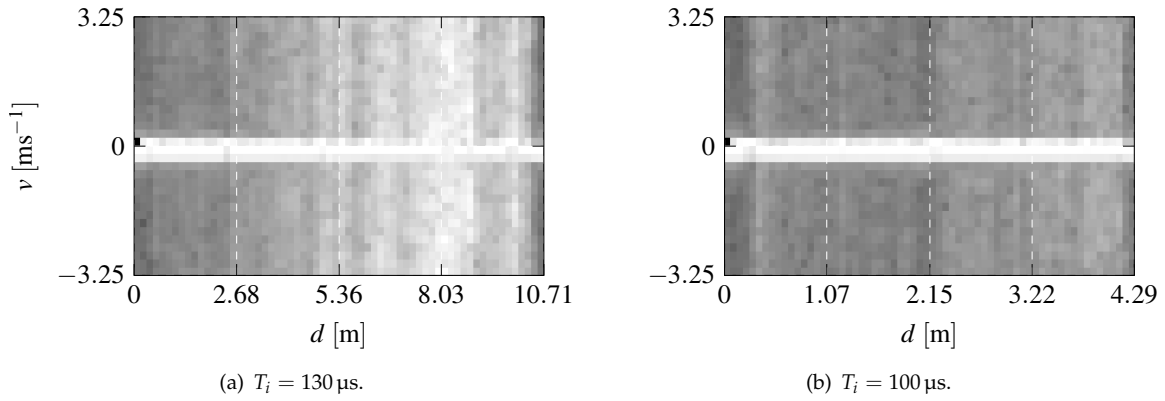
## Range – Doppler heatmap plots

Columns of all the RDH plots are normalized by their maximum value.

### B.1 Adjustment of idle time

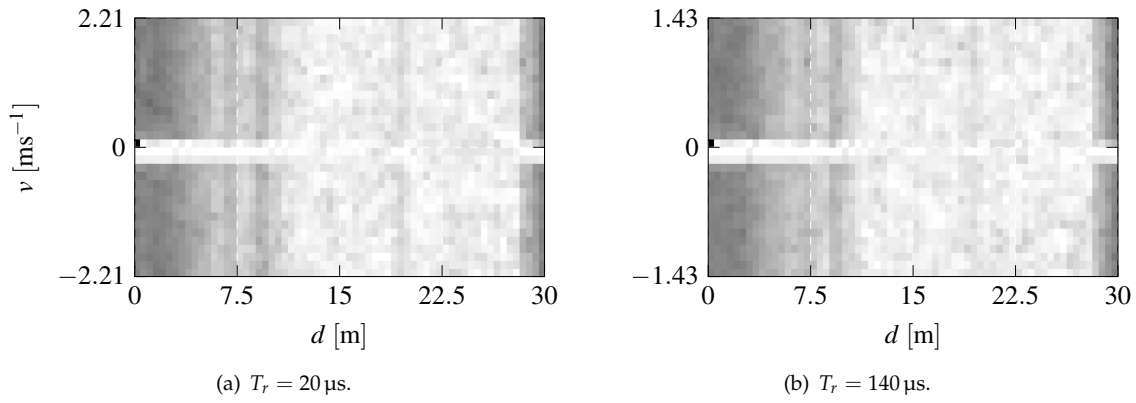


**Fig. B.1:** Range – Doppler heatmap plots for  $f_1 = 77 \text{ GHz}$ ,  $T_{\text{ADC}}^s = 7 \mu\text{s}$ ,  $T_r = 27 \mu\text{s}$ ,  $f_p = 6 \text{ MHz}$ ,  $N = 64$ , and two different  $T_i$  values.



**Fig. B.2:** Range – Doppler heatmap plots for constant  $T_c = 150 \mu\text{s}$ . Here,  $f_1 = 77 \text{ GHz}$ ,  $T_{\text{ADC}}^s = 7 \mu\text{s}$ ,  $T_r|_{T_i=130 \mu\text{s}} = 20 \mu\text{s}$ ,  $T_r|_{T_i=100 \mu\text{s}} = 50 \mu\text{s}$ ,  $f_p|_{T_i=130 \mu\text{s}} = 5 \text{ MHz}$ ,  $f_p|_{T_i=100 \mu\text{s}} = 2 \text{ MHz}$ ,  $N = 64$ .

### B.2 Adjustment of ramp time



**Fig. B.3:** Range – Doppler heatmap plots for  $f_1 = 77 \text{ GHz}$ ,  $T_{\text{ADC}}^s = 7 \mu\text{s}$ ,  $T_i = 200 \mu\text{s}$ ,  $f_p = 5 \text{ MHz}$ ,  $N = 64$ , and two different  $T_r$  values.

## Selected source codes

## C.1 PAPR modeling

```

100 import numpy as np
100 import pandas as pd
100 import re
100 import matplotlib.pyplot as plt
100 from mpl_toolkits import mplot3d
100 from scipy import stats
100 from sklearn.linear_model import LinearRegression
100 from sklearn.preprocessing import PolynomialFeatures
100 from sklearn.pipeline import make_pipeline
100 from sklearn.metrics import r2_score
100 from sklearn.model_selection import train_test_split

101 # ----- Data preprocessing
100 data = pd.read_table(r'measured_data.dat', sep = '\t')
100 data.columns = [re.sub(r'\.[.+\]', '', i) for i in data.columns] # Removing unit brackets
100 data_input = data.loc[:, ['S', 'fs']]
100 data_target = data['papr']

101 # ----- Defining a model
100 model_poly = make_pipeline(PolynomialFeatures(3), LinearRegression())

101 # ----- Calculating accuracy using R^2
100 r_squared = []
100 for c in np.arange(1000):
100     data_input_train, data_input_test, data_target_train, data_target_test = train_test_split(
100         data_input, data_target, test_size=.3)
100     model_poly.fit(data_input_train, data_target_train)
100     papr_modeled = model_poly.predict(data_input_test)
100     r_squared.append(r2_score(data_target_test, papr_modeled))
100 r_squared = np.array(r_squared)

101 print('Median:_{},_IQR:_{},_1st_quartile:_{},_3rd_quartile:_{}'.format(np.median(r_squared3),
100     stats.iqr(r_squared3), np.percentile(r_squared3, 25), np.percentile(r_squared3, 75)))

101 fig, ax = plt.subplots()
100 ax.hist(r_squared, bins=20, density=True, alpha=.5)
100 ax.axvline(np.percentile(r_squared, 50))
100 ax.axvline(np.percentile(r_squared, 5))
100 ax.axvline(np.percentile(r_squared, 95))

101 # ----- Fitting the model to all data and predicting PAPR at parameter space
100 fp = np.linspace(np.min(data['fs']), np.max(data['fs']), num = 100)
100 S = np.linspace(np.min(data['S']), np.max(data['S']), num = 100)
100 fp_mesh, S_mesh = np.meshgrid(fp, S)

```



```

101 model_poly.fit(data_input, data_target)
100 papr_model = model_poly.predict(np.vstack((np.ravel(S_mesh), np.ravel(fp_mesh))).T).reshape(np.
    shape(fp_mesh))

101 # ----- Getting parameters of the model
100 feature_names = model_poly.steps[0][1].get_feature_names(['S', 'fs'])
100 coefficients = model_poly.steps[1][1].coef_
100 intercept = model_poly.steps[1][1].intercept_
100 print('PAPR=0{0+0{0'.format(str(intercept), '0+0'.join(['0*0'.join(str(coefficients[i]),
    feature_names[i])) for i in np.arange(1, len(feature_names))))))

101 # ----- Examining error
100 chi = data_target - model_poly.predict(data_input)
100 fig, ax = plt.subplots()
100 ax.hist(chi, bins=20, density=False)
100 ax.axvline(np.median(chi), color='k')
100 ax.axvline(np.percentile(chi, 1), color='k')
100 ax.axvline(np.percentile(chi, 99), color='k')
100 ax.axvline(np.mean(chi), linestyle='dashed', color='k')

101 chi = chi[chi < np.percentile(chi, 95)]
100 print(stats.kstest(stats.zscore(chi), 'norm'))

101 # ----- Visualizing results
100 fig = plt.figure()
100 ax = fig.add_subplot(111, projection = '3d')
100 ax.plot_surface(fp_mesh, S_mesh, papr_model, cmap = plt.get_cmap('gray'), alpha = .5)
100 sc = ax.scatter(data['fs'], data['S'], data['papr'], c = np.abs(chi), cmap = plt.get_cmap('hot'),
    alpha = 1)
100 ax.set_xlabel('fp')
100 ax.set_ylabel('S')
100 ax.set_zlabel('papr')
100 plt.colorbar(sc)
100 plt.show()

```

**Listing C.1:** Modeling PAPR using linear model with third-order polynomial kernel

## C.2 Other

```

100 import numpy as np
100 import matplotlib.pyplot as plt

101 rax = np.arange(100) # Range axis
100 papr = -.004 * rax + 1.8 + np.random.normal(0, .04, np.size(rax)) # Generating PAPR

101 n_movmean = 10
100 papr_threshold = 1.6

101 movmean = np.convolve(papr, np.ones(n_movmean) / n_movmean, mode='same')
100 movmean[:n_movmean // 2] = np.nan
100 movmean[-n_movmean // 2:] = np.nan

101 # ----- Cut the PAPR vector accordingly to provide an example and plot it
100 papr[np.nanargmin(np.abs(movmean - papr_threshold)):] = np.nan

101 fig, ax = plt.subplots()
100 ax.plot(rax, papr)
100 ax.plot(movmean)
100 plt.show()

```

**Listing C.2:** Selecting maximum range bin to consider according to moving mean of artificial PAPR values.

---

## Acronyms

ADC	.....	Analog-to-digital converter
AOA	.....	Angle of arrival
AOR	.....	Area of reflection
ASCII	.....	American standard code for information interchange
CAN	.....	Controller area network (Standard automotive data interface)
CCS	.....	Code Composer Studio (IDE provided by Texas Instruments)
CLI	.....	Command line interface
CPU	.....	Central processing unit
CUT	.....	Cell under test
CW	.....	Continuous wave
DFE	.....	Digital front-end
DFT	.....	Discrete Fourier transform
DSP	.....	Digital signal processor
EVM	.....	Evaluation module
FFT	.....	Fast Fourier transform (The algorithm for DFT computation)
FMCW	.....	Frequency modulated continuous wave
FPGA	.....	Field-programmable gate array
GPS	.....	Global positioning system
GUI	.....	Graphical user interface
HF	.....	High frequency
HP	.....	High pass
HW	.....	Hardware
IDE	.....	Integrated development environment
IF	.....	Intermediate frequency
IQR	.....	Interquartile range
LED	.....	Light-emitting diode
LP	.....	Low pass (filter)
LSQ	.....	Least squares
MAC	.....	Media Access Control
MIMO	.....	Multiple input, multiple output
MSS	.....	Master subsystem
MUR	.....	Maximum unambiguous range
MWUT	.....	Mann – Whitney <i>U</i> test
NA	.....	Not applicable
NRL	.....	Naval Research Laboratory
PAPR	.....	Peak to average power ratio
PWM	.....	Pulse-width modulation
RDH	.....	Range – Doppler heatmap

RF	Radio frequency
RX	Receiving; Connected with reception
SI	Système international (d'unités) (International system of units)
SOP	Sense-on-power
SSB	Single side band
TI	Texas Instruments (Semiconductor company)
TX	Transmitting; Connected with transmission
UART	Universal asynchronous receiver and transmitter
USD	u.s. dollar (Currency of the United States)
VGA	Variable gain amplifier

---

## Bibliography

- [1] M.A. Richards. *Fundamentals Of Radar Signal Processing*. McGraw-Hill, 2005. ISBN 9780070607378.
- [2] Š. Matějka. *Radarová odometrie. Určení brzdné dráhy vozidla pomocí mikrovlňného dopplerovského radaru (zpráva za rok 2016)*, v. 0.9, 01/2017. Prague, Jan. 2017.
- [3] A. G. Stove. *Linear FMCW radar techniques*. *IEE Proceedings F – Radar and Signal Processing*, 139(5):343–350, Oct 1992. ISSN 0956-375X.
- [4] Andrea Porubiaková and Jozef Komačka. *A Comparison of Dielectric Constants of Various Asphalts Calculated from Time Intervals and Amplitudes*. *Procedia Engineering*, 111:660 – 665, 2015. ISSN 1877-7058. URL <http://www.sciencedirect.com/science/article/pii/S1877705815013788>. XXIV R-S-P seminar, Theoretical Foundation of Civil Engineering (24RSP) (TFoCE 2015).
- [5] C. E. Mungan. *Phase Change upon Reflection*. Available from [https://www.usna.edu/Users/physics/mungan/\\_files/documents/Scholarship/PhaseChange.pdf](https://www.usna.edu/Users/physics/mungan/_files/documents/Scholarship/PhaseChange.pdf). Accessed Jan. 28, 2020.
- [6] V. Pankrác. *Příklady na kolmý dopad vlny: Vlna dopadající na měděný vodič*. Available from [https://elmag.fel.cvut.cz/sites/default/files/users/pankrac/files/vlna\\_na\\_m%C4%9B%C4%8F.pdf](https://elmag.fel.cvut.cz/sites/default/files/users/pankrac/files/vlna_na_m%C4%9B%C4%8F.pdf). Accessed Jan. 28, 2020.
- [7] A.V. Oppenheim, R.W. Schafer, and J.R. Buck. *Discrete-time Signal Processing*. Prentice Hall international editions. Prentice Hall, 1999. ISBN 9780137549207.
- [8] Texas Instruments. *TI mmWave Training*. Available from <https://training.ti.com/mmwave-training-series>. Online training course addressing the software product. Accessed Feb. 2, 2019.
- [9] Texas Instruments. *MIMO radar*, v. 7/2018. Available from <http://www.ti.com/lit/an/swra554a/swra554a.pdf>. Application report. Accessed Jan. 24, 2020.
- [10] Texas Instruments. *AWR1642 Evaluation Module (AWR1642BOOST). Single-Chip mmWave Sensing Solution*, rev. 4/2018. Available from <http://www.ti.com/lit/ug/swru508b/swru508b.pdf>. Reference manual.
- [11] Matej Oravec. *Dopplerovské měření rychlosti pro systém určování polohy*. Bachelor's thesis, CTU in Prague, Prague, May 2018.
- [12] C. Rusch, T. Klein, and S. et al. Beer. *A Short Distance CW-Radar Sensor at 77 GHz in LTCC for Industrial Applications*. *J Infrared Milli Terahz Waves*, 34:856–865, 2013.

- [13] R. I. Cojocaru, E. Moldovan, B. Boukari, S. Affes, and S. O. Tatu. *A new 77 GHz automotive phase coded CW multi-port radar sensor architecture*. In *2008 European Radar Conference*, pages 164–167. IEEE, Oct 2008. ISSN null.
- [14] RFbeam Microwave GmbH. *MR3003\_RD Radar Transceiver*. Available from <https://www.rfbeam.ch/product?id=27>. Accessed Feb. 04, 2020.
- [15] NXP. *S32R27 Automotive Radar Reference Design Kit*. Available from <https://www.nxp.com/design/development-boards/automotive-development-platforms/s32r-mcu-platforms/s32r27-reference-design-kit-for-high-performance-automotive-radar:RDK-S32R274>. Accessed Feb. 03, 2020.
- [16] Texas Instruments. *AWR1642 Single-Chip 77- and 79-GHz FMCW Radar Sensor*, rev. 4/2018. Available from <http://www.ti.com/lit/ds/symlink/awr1642.pdf>. Reference manual. Accessed Jan. 24, 2020.
- [17] E. Holzman. *On the use of dummy elements to match edge elements in transmit arrays*. In *2013 IEEE International Symposium on Phased Array Systems and Technology*, pages 549–552. IEEE, 2013.
- [18] N. Naveen. *AWR1642 Bootloader Flow*. Available from <http://www.ti.com/lit/an/swra551/swra551.pdf>. Accessed Feb. 06, 2020.
- [19] P. Pechač and S. Zvánovec. *Základy Šíření vln pro plánování pozemních rádiových spojů*. BEN – Technická literatura, 2007. ISBN 978-80-7300-223-7.
- [20] N. M. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer, 2002. ISBN 978-80-7300-223-7.
- [21] *Conic Section of an Elliptical Cone*. Available from <http://mathforum.org/library/drmath/view/72799.html>. Accessed Jan. 4, 2020.
- [22] Texas Instruments. *Millimeter Wave (mmw) Demo for XWR16XX*. Available from the documentation provided with TI MMWaveSDK software. The software is available from [http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02\\_01\\_00\\_04/index\\_FDS.html](http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02_01_00_04/index_FDS.html). Documentation.
- [23] A. A. Pirhani, S. Pooni, and M. Cherniakov. *Implementation of MIMO Beamforming on an OTS FMCW Automotive Radar*. In *2019 20th International Radar Symposium (IRS)*, pages 1–8. IEEE, June 2019. ISSN 2155-5745.
- [24] *Raspberry Pi 3 Model B+*. Available from <https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>. Accessed Feb. 19, 2020.
- [25] Sony Energy Devices Corporation. *US18650VTC6 Lithium Ion Rechargeable Battery Technical Information*. Available from [https://www.kronium.cz/uploads/SONY\\_US18650VTC6.pdf](https://www.kronium.cz/uploads/SONY_US18650VTC6.pdf). Accessed Feb. 19, 2020.
- [26] *LM2596*. Available from <http://datasheet.hezkyden.cz/d/lm2596-on.pdf>. Datasheet. Accessed Feb. 19, 2020.
- [27] Texas Instruments. *TI Resource Explorer: Industrial Toolbox*. Available from [http://dev.ti.com/tirex/explore/node?node=AJ0MGA2ID9pCPWEKPi16wg\\_\\_VLYFKFf\\_\\_LATEST](http://dev.ti.com/tirex/explore/node?node=AJ0MGA2ID9pCPWEKPi16wg__VLYFKFf__LATEST). Accessed Feb. 11, 2020.
- [28] Texas Instruments. *Uniflash Standalone Flash Tool for TI Microcontrollers (MCU), Sitara Processors & SimpleLink devices*. Available from <http://www.ti.com/tool/UNIFLASH>. Accessed Feb. 11, 2020.
- [29] Texas Instruments. *Code Composer Studio (CCS) Integrated Development Environment (IDE)*. Available from <http://www.ti.com/tool/CCSTUDIO>. Accessed Feb. 12, 2020.

- [30] Texas Instruments. *TI Resource Explorer*. Available from [http://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg\\_\\_VLyFKFf\\_\\_LATEST](http://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg__VLyFKFf__LATEST). Accessed Feb. 12, 2020.
- [31] Texas Instruments. *MMWAVE SDK User Guide*, v.1.0 (10/2018). Available from [http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02\\_01\\_00\\_04/index\\_FDS.html](http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02_01_00_04/index_FDS.html). Software product documentation. Accessed: Jan. 24, 2020.
- [32] V. Dham. *Programming Chirp Parameters in TI Radar Devices*, v. 5/2017, rev. 2/2020. Available from <http://www.ti.com/lit/an/swra553a/swra553a.pdf>. Application report. Accessed Feb. 25, 2020.
- [33] K. Ramasubramanian. *Using a complex-baseband architecture in FMCW radar systems*. Available from <http://www.ti.com/lit/wp/spyy007/spyy007.pdf>. Accessed Feb. 13, 2020.
- [34] F.E. Nathanson, J.P. Reilly, and M.N. Cohen. *Radar Design Principles: Signal Processing and the Environment*. Scitech Pub., 1999. ISBN 9781891121098.
- [35] M. A. Abdel-Nabi, K. G. Seddik, and E. A. El-Badawy. *Spiky sea clutter and constant false alarm rate processing in high-resolution maritime radar systems*. In *2012 International Conference on Computer and Communication Engineering (ICCCCE)*, pages 478–485. IEEE, July 2012.
- [36] J. Sýkora. *Teorie digitální komunikace*. Praha, ČVUT, first edition, 2002. ISBN 80-01-02478-4.
- [37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python*. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [38] *Comments in rL\_sensor.h source code*. Available from [http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02\\_01\\_00\\_04/index\\_FDS.html](http://software-dl.ti.com/ra-processors/esd/MMWAVE-SDK/02_01_00_04/index_FDS.html). Part of *mmwavelink* software in *mmWave SDK 2.0.0.4*.
- [39] Nadim Nachar. *The Mann-Whitney U: A Test for Assessing Whether Two Independent Samples Come from the Same Distribution*. *Tutorials in Quantitative Methods for Psychology*, 4, 03 2008.
- [40] Ch. J. Wild. *The Wilcoxon Rank-Sum Test*. Online. Available from <https://www.stat.auckland.ac.nz/~wild/ChanceEnc/Ch10.wilcoxon.pdf> [Mar. 26, 2020], 1997. Supplement for Chapter 10 of the book “Wild, Ch. J. and Seber, G. A. F. *Chance Encounters: A First Course in Data Analysis and Inference*. New York: Wiley, 1999. ISBN: 978-0-471-32936-7”.
- [41] I. Hajnsek and K. Papathanassiou. *Rough surface scattering models*. Available from [https://earth.esa.int/documents/653194/656796/Rough\\_Surface\\_Scattering\\_Models.pdf](https://earth.esa.int/documents/653194/656796/Rough_Surface_Scattering_Models.pdf) [Apr. 01, 2020], 2005.
- [42] *Locus Map website*. Available from <https://www.locusmap.eu/>.
- [43] *GPX: the GPS Exchange Format*. Available from <https://www.topografix.com/gpx.asp>. Specification of the GPS Exchange Format.
- [44] Python foundation. *strftime() and strptime() Format Codes*. Available from <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>. Accessed Apr. 17, 2020.
- [45] *gpxpy – GPX file parser*. Available from <https://pypi.org/project/gpxpy/>.
- [46] P. Welch. *The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms*. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967.
- [47] *dss\_main.c source code of the mmWave Demo firmware*. Part of the *Industrial Toolbox* v. 3.6.0. Available from [http://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg\\_\\_VLyFKFf\\_\\_LATEST](http://dev.ti.com/tirex/explore/node?node=AJoMGA2ID9pCPWEKPi16wg__VLyFKFf__LATEST).

- [48] Texas Instruments. *mmWave Demo Visualizer*, v. may 2017, rev. oct 2018. Available from <http://www.ti.com/lit/ug/swru529b/swru529b.pdf>. User's Guide. Accessed Mar. 20, 2020.