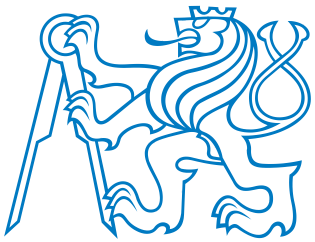
Czech Technical University in Prague

Department of Electromagnetic Field



BACHELOR THESIS

Surveillance FMCW Radar

Supervisor of the bachelor thesis: Ing. Viktor Adler, Ph.D

Study programme: Elektronika a komunikace

Prague May 2024

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

Já, níže podepsaný

Student's surname, first name: Havránek Kryštof
Personal number: 516206
Program name: Elektronika a komunikace

prohlašuji, že jsem bakalářskou práci s názvem

Přehledový FMCW radar

vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu.

Prohlašuji, že jsem v průběhu příprav a psaní závěrečné práce použil nástroje umělé inteligence. Vygenerovaný obsah jsem ověřil. Stvrzuji, že jsem si vědom, že za obsah závěrečné práce plně zodpovídám.

V Praze dne 12.05.2025

Kryštof Havránek
.....
podpis

Acknowledgements
TODO

Title: Surveillance FMCW Radar
Author: Havránek Kryštof
Department: Department of Electromagnetic Field
Supervisor: Ing. Viktor Adler, Ph.D, Department of Electromagnetic Field

Abstract: TODO

Keywords: FMCW, Surveillance Radar, ESP32, MATLAB, Digital Signal Processing, 3D Printing, G-code

Název práce: Přehledový FMCW radar
Autor: Havránek Kryštof
Katedra: Katedra elektromagnetického pole
Vedoucí práce: Ing. Viktor Adler, Ph.D, Katedra elektromagnetického pole

Abstrakt: TODO

Klíčová slova: FMCW, Přehledový rdar, ESP32, MATLAB, Číselné zpracování signálů, 3D tisk, G-code

Contents

Abbreviations

Abbreviation	Meaning
CFAR	Constant False Alarm Rate
AGC	Automatic Gain Control
CW	Continuous Wave
devkit	Development Kit
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DFT	Discrete Fourier Transform
FMCW	Frequency Modulated Continuous Wave
FFT	Fast Fourier Transform
MIMO	Multiple Input Multiple Output
SNR	Signal to Noise Ratio
SISO	Single Input Single Output
TSV	Tab Separated Values
RCS	Radar Cross Section

Introduction

Following theses concerns itself with realization of a generic surveillance radar based on FMCW technology. Commonly surveillance radars based on FMCW technology either have multiple receiving antennas or are realized as MIMO systems. In case of MIMO systems electronic steering of the radar beam is possible, allowing for a larger field of view and target detection both in elevation and azimuth. For simpler systems with just multiple receiving antennas only azimuth estimation is possible based on the phase difference analysis [1]. These systems however usually require more processing power and aren't able to provide information about whole 3D space.

Therefore a simpler solution is proposed with radar system that has only one RX and TX antenna. System is then steered mechanically using a rotary platform. At first capabilities of SiRad Easy© evaluation board are analyzed and then a custom rotary platform is constructed. Both are then integrated into a single system using MATLAB desktop application. This involves both hardware, in a form of two axis rotary platform, and software development be it for driving the rotary platform or processing the radar data. Similar systems have been already realized but relied only one axis of rotation [2, 3].

FMCW is a radar technology that uses a continuous wave signal whose frequency is modulated in time. By mixing sent signal over received a harmonic components are created whose frequencies are proportional to the distance of the target [4]. Thus by applying a Fourier transform to the received signal, distances of object in the scene can be determined. As opposed to pulsed radar, FMCW is able provide accurate distance measurement with relatively low power consumption [5]. Speed estimations of the target is also possible by using the Doppler effect.

As already stated this thesis is focused on the SiRad Easy© radar system. It is a low-cost evaluation board designed to familiarize users with the FMCW technology offering both 24 GHz header and 122 GHz [6]. This allows a relatively larger range of applications with ranges from couple of meters to up to 400 m under ideal condition [7]. However as SiRad Easy© offers rather low sampling rate and uses relatively low speed microcontroller it is not really suitable for speed measurement. Maximal speeds detectable are in the ranges of tens of mm per second and even then are not very accurate.

For the purpose of this thesis a custom rotary platform was designed and constructed. While such platforms already exists [8, 9] most of them are prohibitively expensive and offer unnecessary features. Thus a custom solution was designed using off-the-shelf and 3D printed parts. Whole system is then controlled with ESP32C6 microcontroller which interpreters G-code like commands and drives the stepper motors.

Data from the radar and the platform then are processed and visualized using MATLAB. Processing pipeline is relatively standard and uses common techniques such as FFT (Fast Fourier Transform) and CFAR (Constant False Alarm Rate) [10]. However a number of steps can be turned on and off, or their behavior modified by the user, allowing a great deal of flexibility. The data are then stored in radar cubes, common structure used in such applications [10]. This storage format allows implementation of more complex post processing algorithms. Pro-

cessed data can then be visualized in a number of ways depending on user needs – from simple 2D range-azimuth plots to visualization of the whole 3D space.

Thesis is split into five main chapters. First provides theoretical background to FMCW radar workings and its advantages over other possible radar technologies. Second chapter is focused on SiRad Easy© evaluation board and its capabilities specifically in relation to its application in surveillance radar. Third chapter describes the design requirements and the design process of custom rotary platform. Software needed to drive the rotary platform is also described in this chapter. Fourth chapter is a general overview of the MATLAB desktop application used to control the whole system and process the radar data. With fitch chapter focusing on the data processing pipeline and visualizations in depth.

1. FMCW Radar Fundamentals

While classical Continuous Wave (CW) radars broadcast a signal only on a single frequency, Frequency-Modulated Continuous Wave (FMCW) radars make a linear sweep across a given bandwidth. This approach enables accurate range estimation at low power consumption and still allows speed measurements using the Doppler shift. However, velocity calculations in FMCW radars are more complex compared to those in single-frequency CW radars.

The "MW" suffix in FMCW radar denotes that the system operates in the microwave frequency range. These high frequencies enable the use of compact antenna arrays, which can sometimes even be integrated directly on-chip. Additionally, the millimeter-wave portion of the spectrum is typically license-free [11] and offers large bandwidths, reducing a risk of interference.

1.1 Comparison FMCW Radar to Pulse Radar

Distance measurement using radar predates FMCW technology by several decades. Early radar systems primarily relied on pulsed signals, where distance can be calculated from the time of flight, t, of the signal as

$$d = \frac{c_o \cdot t}{2}. \tag{1.1}$$

Speed calculation is also relatively straightforward, as the Doppler effect is quite clear

$$v = \frac{f_{\text{dop}} c_o}{2 f_{\text{rad}}}, \tag{1.2}$$

where f_{dop} is the Doppler frequency, c_o is the speed of light and f_{rad} is the frequency of the radar signal [5].

While this approach is conceptually straightforward, it has several limitations, particularly in applications requiring high precision at close range. Achieving fine resolution in distance measurement necessitates very short pulses. However, to maintain sufficient signal-to-noise ratio (SNR), the transmitted pulse power per pulse must remain high, regardless of its width [5].

Maintaining high average transmission power poses legal and technical challenges. Risk of interference with other devices is increased and such system demand bulky, high-power circuitry – often with high voltages and even vacuum tubes. Consequently, pulsed radar is predominantly used in applications where fine range resolution is not essential, such as long-range target detection.

One key advantage of pulsed radar is its relatively simple data processing. In contrast, FMCW radar data processing is more complex due to the interdependence of distance and velocity measurements. Both distance and velocity of a target cause frequency changes in the received signal.

1.2 Basic principles of ideal FMCW radar

Let us picture an ideal FMCW radar system placed in a scene with a single large target that reflects all incoming signals. The radar is broadcasting with a

periodically repeated frequency sweep from f_c to $f_c + BW$ creating a sawtooth waveform. Aside from a sawtooth, other modulations are also frequently used, such as linear triangular modulation or segmented linear frequency modulation. These offer some advantages, but the nature of the beat signal, which forms a sine wave with sawtooth modulation, is more complex. This is especially true in the case of triangle modulations when multiple targets are present [5]. SiRad Easy© kit, used in this thesis, technically uses a segmented linear frequency modulation due to its limited computational power [12], but the mathematical principles are similar to those for sawtooth.

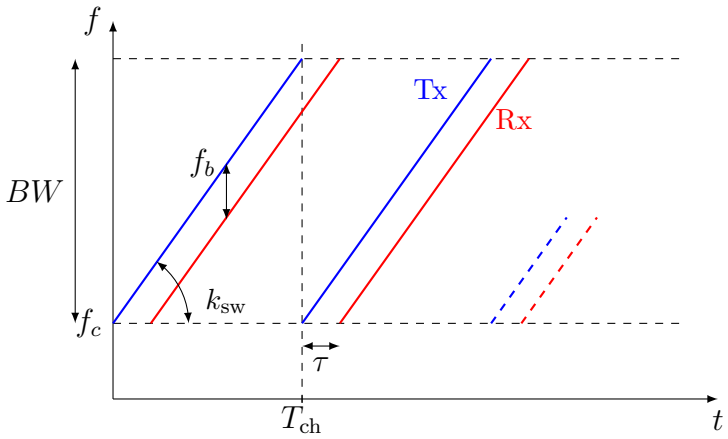


Figure 1.1: Ideal relation of frequency on time for received and sent signal [13]

We can clearly see that, in a given time t, the frequency spread from the sent signal to the received signal is proportional to the time delay τ. However, performing a simple subtraction in the spectrogram of both signals isn't really feasible; the calculation needs to take a smarter approach.

Let us define the chirp slope k_{sw} with which we can describe the change in frequency of the broadcasted signal as

$$\Delta f_{\mathrm{s}}(t) = k_{\mathrm{sw}} t = \frac{BW}{T_{\mathrm{ch}}} t, \tag{1.3}$$

where t is the time, ranging 0 to chirp length T_{ch}. The standard equation of an frequency modulated signal can be written as

$$s_{\mathrm{t}}(t) = A \cos\left(\omega_c t + 2\pi \int_0^t f_{\mathrm{s}}(t)\mathrm{d}t\right), \tag{1.4}$$

where A is the amplitude of the signal, ω_c is the carrier frequency and $f(s)$ is the frequency of the signal. Substituting (1.3) into (1.4) we get the signal broadcasted by the radar [14]

$$s_{\mathrm{t}}(t) = A \cos(\omega_c t + \pi k_{\mathrm{sw}} t^2). \tag{1.5}$$

The signal bounced back from a target will have the same equation, only shifted by the time delay τ,

$$s_{\mathrm{r}}(t) = A \cos(\omega_c (t - \tau) + \pi k_{\mathrm{sw}} (t - \tau)^2). \tag{1.6}$$

Now we can calculate the product of the two signals; this can be done easily in the real world using a frequency mixer. The result of the multiplication is [14]

$$s(t) = s_{\mathrm{r}}(t) \cdot s_{\mathrm{t}}(t) = \frac{A^2}{2} \cos\left(2(\omega_c - 2\pi k_{\mathrm{sw}} T_\tau)t + 2\pi k_{sw} t^2 + (\pi k_{\mathrm{sw}} \tau^2 - \omega_c \tau)\right) + \\ + \frac{A^2}{2} \cos\left(2\pi k_{\mathrm{sw}} \tau t + (\omega_c \tau - \pi k_{\mathrm{sw}} \tau^2)\right). \tag{1.7}$$

The first additive term will lead to a signal with a very high frequency, well above $2\omega_c$. This term doesn't carry any useful information and is usually filtered out – either by a low-pass filter or the frequency mixer itself. The second term is the so-called beat signal, whose frequency is directly proportional to the time delay τ. Deriving the second argument by time, we get the frequency of the beat signal [4]

$$f_{\mathrm{b}} = \frac{1}{2\pi} \frac{\partial}{\partial t} \left(2\pi k_{\mathrm{sw}} \tau t + (\omega_c \tau - \pi k_{\mathrm{sw}} \tau^2)\right) = k_{\mathrm{sw}} \tau. \tag{1.8}$$

Calculating the distance to a target is now trivial as the delay τ is equal to the time it takes for the signal to travel to a target and back [4]

$$R = \frac{c_0 \tau}{2}, \tag{1.9}$$

By substituting (1.8) into (1.9), we get the equation for a distance [4]

$$R = \frac{c_0 f_{\mathrm{b}}}{2k_{\mathrm{sw}}} = \frac{c_o f_{\mathrm{b}} T_{\mathrm{ch}}}{2BW}. \tag{1.10}$$

1.2.1 Limits of Range Measurement

The maximal distance is given by the time it takes for the signal to travel from the radar to a target and back. If the distance were greater than the time of a single chirp, the signal would be interpreted as coming from a closer target. This gives us a maximal limit on the beat frequency, $f_{\mathrm{b}} = BW$.

However, in most cases, the limit will be imposed not by T_{ch} respectively BW but by the sampling frequency f_{s}. In order to avoid aliasing, the Nyquist-Shannon theorem must be satisfied, thus limiting the maximal beat frequency to $f_{\mathrm{s}}/2$ and resulting in a maximal distance of [5]

$$R = \frac{c_o f_{\mathrm{s}}}{4k_{\mathrm{sw}}}. \tag{1.11}$$

While sampling with frequency f_{s} we get $N = \mathrm{f_s} T_{\mathrm{ch}}$ samples. Applying a DFT to the signal we get N samples in spectrum with frequency resolution of

$$\Delta f_{\mathrm{b}} = \frac{f_{\mathrm{s}}}{N} = \frac{1}{T_{ch}}. \tag{1.12}$$

We can see that the resolution of the spectrum is only inversely proportional to the chirp length and doesn't have any relation to the sampling frequency [5]. Now we can substitute Δf_{b} into (1.10) to get the minimal distance that can be measured as

$$\Delta R = \frac{c_o}{2BW} \tag{1.13}$$

Thus, in order to increase resolution in range, a wider bandwidth is needed [4].

There are, of course, other effects impeding the resolution of the radar system, such as phase noise around targets or sweep nonlinearity. Sweep nonlinearity can be either in the ramp itself, leading to decreasing resolution with range (with both linear and quadratic errors present) [4], or in sweep recovery (time to return to the start of the sweep), which leads to a constant decrease in resolution [15]. Both are, however, largely compensated in modern radar systems by using a closed feedback loop [4].

1.2.2 Speed Measurement

In order to demonstrate the effect of moving target on the beat frequency, we can redefine the time delay τ as

$$\tau = \frac{2(R_0 + vt)}{c_o} \tag{1.14}$$

where R_0 is the initial distance to a target and v is the radial speed of a target. Within a single chirp, there is no way to distinguish between the effects of distance and speed of the target; thus, multiple chirps are needed [14]. We can rewrite the equation (1.14) as

$$\tau = \frac{2(R_0 v(nT_{\text{ch}} + t_{\text{s}}))}{c_0} \tag{1.15}$$

where n is the number of chirps, T_{ch} is the chirp length (or in case of segmented signal whole cycle duration) and t_{s} denotes time within a single chirp ($0 \leqslant t_{\text{s}} \leqslant T_{\text{ch}}$). Substituting (1.15) into low-frequency part (1.7) leads to a very complex equation. However, according to [14], most of its terms can be neglected, with result being

$$s(t_{\text{s}}, n) = \frac{A^2}{2} \cos\left(\frac{4\pi k_{\text{sw}} R_0}{c_0} t_{\text{s}} + \frac{2\omega_c v n}{c_0} T_{\text{ch}} + \varphi_0\right), \tag{1.16}$$

where φ_0 is a phase shift given by the initial distance to a target. It is clear that the first element describes predominantly the distance to a target, and the second one describes the speed of a target. We can also see that speed will not affect the beat frequency in a single spectrum but will lead to a phase shift across multiple spectra [14].

In order to calculate the Doppler shift frequency,

$$f_{\text{d}} = \frac{2 f_{\text{c}} v}{c_0}, \tag{1.17}$$

where f_c is the center frequency, a 2D Fourier transform can be used. First, an FFT is run on each individual chirp, and then another FFT is applied to these individual outputs in order to determine the phase shift. Given that the FFT spectrum is calculated from complex data, both halves of the spectrum contain useful information. After applying `fftshift`, we get a spectrum from negative radial speeds (moving away from the radar) to positive (moving towards the radar) with zero in the middle. This will lead to a so-called range-Doppler map, which on one axis contains information about speed and on the second axis contains information about the distance of the target.

Speed resolution is derived from the number of chirps N we are analyzing and their length T_{ch} as [14]

$$\Delta v = \frac{c_0}{2 f_c} \frac{1}{N T_{\text{ch}}}. \tag{1.18}$$

The maximal speed is dictated by how large of a phase shift we can measure without ambiguity, that is $\pm\pi$. From (1.16), we can formulate the maximal speed measurable as

$$v_{\max} = \frac{c_0}{4 f_c T_{\text{ch}}},$$

according to [16]. But given the nature of FFT, this speed applies to the negative side of the spectrum – that is, for targets moving away from the radar. In the case where the target is moving towards the radar, the speed limit is smaller by a single speed bin width

1.3 Radar equation

Fundamental to all radar systems is the so-called radar equation. This formula establishes a relation between the transmitted/received power, the Radar Cross Section (RCS) of the target, and other system parameters to the maximal range of the radar. It can be written as

$$R_{\max}^4 = \left(\frac{P_{\text{CW}} G_{\text{T}} G_{\text{R}} \lambda^2}{(4\pi)^3 L k T F_{\text{R}} B_{\text{Ro}} (SNR_{\text{Ro}})} \sigma_T \right), \tag{1.19}$$

where P_{CW} is the average system power in watts, G_{T} and G_{R} gain of transmitting and receiving antennas, σ_T is the radar cross section of a target, λ is wavelength, L is the system loss, k is the Boltzmann's constant, T is the effective temperature of the system in Kelvin, F_{R} is the receiver noise figure, B_{Ro} is the receiver bandwidth and SNR_{Ro} is required SNR [5].

Knowing all parameters, or calibrating the system based on a known reference, we can calculate the target's RCS. This value gives us a much clearer understanding of the target as opposed to just displaying the power received or its logarithm. This is due to RCS being a constant value for a given target and its orientation, regardless of the distance [10].

1.4 Angle of Arrival/Spatial Information

Commonly, in order to get spatial information, radars rely on multiple RX antennas. This allows for the estimation of the angle of arrival (AoA) of the target by analyzing phase differences between signals received on different antennas [14]. However, usually, only a single linear series of RX antennas is employed; therefore, only one angle can be estimated.

More complex MIMO systems employ not only multiple RX antennas but also multiple TX antennas. Multiple TX antennas enable beamforming, which is the process of steering the radar beam in a specific direction using positive interference. If antennas are still arranged in a linear array, this only makes the angle of estimation more accurate by effectively creating virtual RX antennas. However, if a 2D arrangement of antennas is used, the system can estimate both azimuth and elevation angles [1].

While these systems allow for superb performance, they are also very expensive and require a lot of processing power. Therefore, this thesis works with a simpler and older solution: a system with a single RX antenna and TX antenna, where the radar is pointed in a given direction mechanically. This approach is

not only cheaper but also allows for the gathering of information about the whole 3D space with just a single radar.

1.5 CFAR

CFAR is a simple algorithm that enables basic target detection in noisy radar data. For each range bin, its neighborhood is analyzed, and the average power level is calculated. This neighborhood is separated by a number of guard cells from the cell under test, which do not partake in the calculation. If the bin under test is greater than the scaled average power level of the neighborhood, we can assume that there is a target present [5].

There are many different variations of CFAR, each with its own advantages and disadvantages. What was described is a basic cell averaging CFAR (CA-CFAR). This algorithm works best in homogeneous environments, where the noise level is relatively constant across the neighborhood. In cases where this cannot be assumed, for example, CFAR with the greatest of the neighborhood (CAGO-CFAR) can be used. This algorithm works by taking the maximum value of the neighborhood and comparing it to the bin under test instead of the mean [17].

2. SiRad Easy©

Indie Semiconductor's SiRad Easy© is an FMCW radar system development kit designed primarily for automotive applications. Out of the box, it offers two headers a 24 GHz and a 122 GHz both based on ICs from Indie Semiconductor (TRX-024-007 and TRA-120-001, respectively). Both are strictly SISO ICs with two antennas – one for receiving another for transmitting so azimuth estimation from phase shift is not possible.

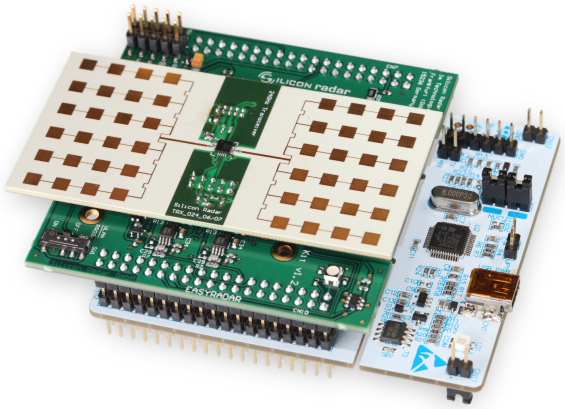


Figure 2.1: SiRad Easy© 24 GHz configuration [7]

Direct communication with the radar header is not possible, or at least, the communication interface is undocumented. Instand there is always and intermediary in the form of STM32 Nucleo series microcontroller. To this microcontroller does the user connect either directly with UART over USB or with WiFi enabled by on-board ESP32. Both are relatively low bandwidth communication – the serial maxing at baudrate of 1 000 000 [6].

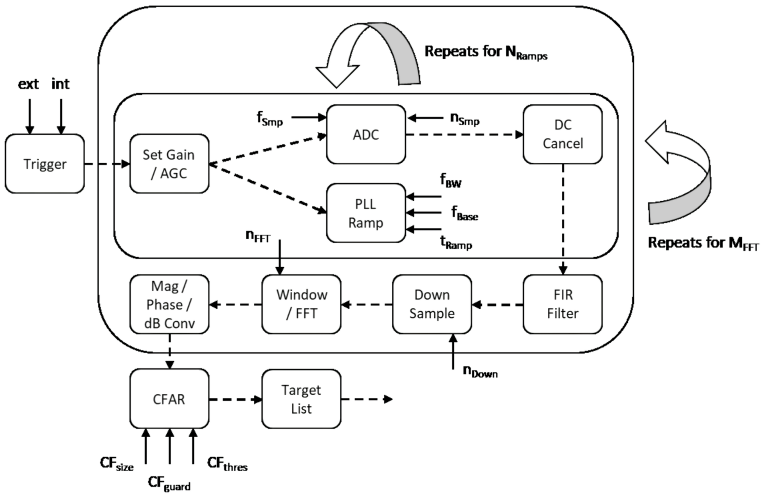


Figure 2.2: Flow of Radar Measurement on SiRad Easy© [12]

The devkit is designed for easy integration into existing projects; however, this unfortunately poses limitations for this thesis. In normal operation, the radar system functions as a black box (processing schema shown in 2.2), implementing

its own filtering, FFT, and CFAR algorithms while only reporting a target list to the user.

This entire sequence can be triggered either internally (driven by a configurable oscillator) or externally (via a GPIO pin or UART input). Once trigger is received the device carries out a user configured number of chirps. The chirp length is determined by the number of sampling steps and the ADC sampling time as follows:

$$t_{\text{ramp}} = \frac{t_{\text{ADC}} \cdot (N_{\text{samples}} + 85)}{36 \text{ MHz}} \ \mu\text{s}, \tag{2.1}$$

with the manufacturer recommending an optimal time of 1 ms for good SNR [12].

SiRad is design to leverage coherent averaging to improve SNR of the output signal where multiple ramps are broadcasted and averaged. While this technique was found to significantly aid in measurement in a cluttered environments averaging effectively destroy the information about the Doppler shift. Additionally, due to the devkit's limited computational resources instead of sawtooth waveform radars broadcasts a segmented one with significant delays in between[12].

The devkit is also clearly designed primarily for range measurement with little to no regard payed for speed calculation. Not only are chirps spaced with large delays (at minimum some 20 ms) but the radar doesn't even maintain constant intervals between chirps. This is the case even when relying on manual triggering of the radar, when using radar's self trigger results are even worse.

When measuring reporting interval with 20 ms trigger the radar achieved average time between updates of 20.5 ms with standard deviation of 1.3 ms. However from time to time they were outliers when report came in only after 30 ms in the worst case or just 16 ms in the best case. This was on relatively small dataset of 5000 samples. In order to rule out that the problem isn't only in reporting output signal from the radar was captured by Rohde & Schwarz FSW26 Signal and Spectrum Analyzer. By analyzing some 40 samples, measured with the same trigger interval of 20 ms, average time between chirps was 20.7 ms with standard deviation of 0.7876 ms thus proving broadcast itself is irregular.

Unfortunately as the radar doesn't maintain constant intervals between chirps speed measurement precision is severely limited. This could be partially mitigated if we knew when the chirp was broadcasted, giving us ability to introduce a correction. However no such information is provided by the radar. Thus each chip is timestamp with it's time of arrival which is assumed to be the time of chirp broadcast. Respectively broadcast took time a trigger period before but this shift is irrelevant to the outcome. Using this timing information speed spectrum can still be estimated using non-uniform DFT.

Per radars manual [12] following equation was used to calculate width of a single range bin

$$\Delta R = \frac{c_0(N + 85)}{2 \cdot BW \cdot NFFT}, \tag{2.2}$$

where variable N denotes number of samples and $NFFT$ number of DFT points used to calculate the spectrum. Except the corrective constant of 85 it closely follows the formula (1.10).

As for velocity the manufacturer doesn't recommend any formula so (1.18) was used with T_{ch} substituted by user-chosen trigger period. However as the minimal period of chirps is still rather high not to mention the radar doesn't maintain constant intervals between chirps, the velocity measurement is not very

practical. Were we to use 24 GHz header with 25 ms trigger we get maximal speed of 0.124 m/s and on low 8 NFFT steps the resolution is only 0.031 ms^{-1}.

This lack of perforace severely limits the radar's usability with regard to surveillance applications. Most surveillance radars are designed to monitor dynamic environments, where targets is moving. And usually at much higher speeds than the SiRad Easy© can measure. This radar is completely unsuitable for such applications and is mainly applicable to static environments. For example at parking lot, operator can detect if a car is parked in a given spot or not. However beside such binary information about presence of a target and its distance not much more can be said.

2.1 Outline of the Chosen Configuration

For the purpose of this project, the devkit was configured to output raw data from the 12-bit ADC in the form of in-phase and quadrature components of the signal. This allows for a more complex processing than if spectrum was calculated on the radar itself. For these reasons, the default windowing and signal filtering were also disabled. Device was set up to be triggered by command sent over UART.

Another parameter to consider in relation to surveillance application is AGC. Using it adds two additional ramps that are used solely to set gain value leading to some 3 ms delay. However bigger problem is the fact that radar doesn't report the gain value. Using AGC could then lead to situations where weighting of the data is not constant during operation. Thus it was deemed safest to turn AGC off and set the gain manually depending on how the readings appear after processing.

It is also important to note that the radar system is not well-suited for on-the-fly configuration changes. Applying a new configuration takes some time and the radar does not provide any feedback to indicate when a new configuration was applied. As a result, commonly used techniques such as alternating chirp slope are not feasible.

In regards to the output format, the radar system supports two options: binary and TSV. Since the output speed does not differ significantly between the two, the primary deciding factor was ease and speed of parsing. Even when MATLAB is not particularly optimized for parsing binary data, it still processes binary output approximately 40% faster than human-readable TSV data.

Other parameters such as sampling frequency, number of samples and so on are left to the user to configure. Setting these strongly depends on what header is used and what environment is the radar used in. Thus making their configuration static wouldn't be a good idea.

2.2 24 GHz Header

Center of 24 GHz header is a SISO TRX-024-007 transceiver which integrates low noise amplifier, frequency mixer, filters and VCO into a single chip. It is primarily designed to operate in the ISM band (24.0–24.25GHz), but also support an additional ultra-wideband mode with frequency range from 23 GHz to 26 GHz [18]. On the SiRad Easy© there is no distinction made between those two modes and the user is free to set any bandwidth [12] . The transmitter output power ranges from 2.5 dBm to 6 dBm, depending on the configuration [18]. A maximum

range of 400 m is advertised [7], though this is likely under ideal conditions when observing a large target.

As shown in figure 2.1, the chip is connected to two microstrip patch antennas. The patches are arranged in a relatively standard configuration, forming a 6×4 array with spacing approximately equal to half the wavelength at 24 GHz.

Since the manufacturer did not provide any information about the radiation pattern of the array, a simulation was conducted using the CST simulation suite. Additionally, the manufacturer did not disclose the substrate parameters for the radar board specifically. Fortunately, the TRX-024-007 datasheet includes a board stack-up for the chip's evaluation board. It was assumed that the same stack-up would be used – 18 μm copper for the traces and ground plane, with a 250 μm thick Rogers RO4350B substrate [18].

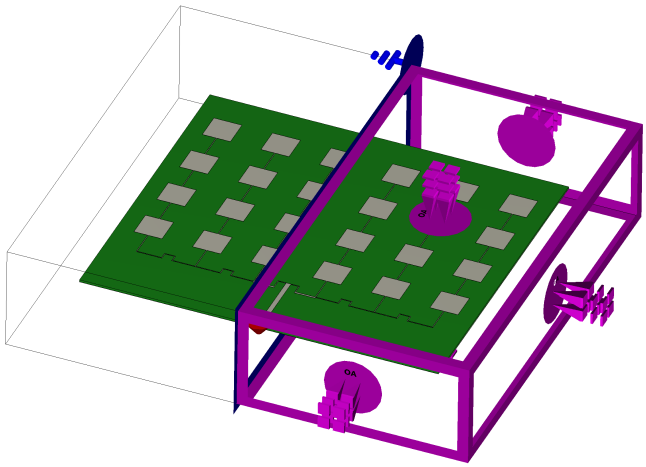


Figure 2.3: Simulated 24 GHz header with boundary conditions showed

After measuring the dimensions of the array using an optical microscope, the entire array was redrawn in CST Studio. Figure 2.3 shows the applied boundary conditions: the antenna was placed in open space, and an $H = 0$ condition was set along the symmetry plane to speed up the simulation.

After performing a standard time-domain simulation with an excitation signal ranging from 0 to 26GHz, the antenna array exhibited a minimum reflection coefficient $s_{11} \doteq -26.4$ dB at 23.478 GHz with second minimum at 24.518 GHz (Figure 2.4). Lack of minimum at 24 GHz may be attributed to difficulties measuring trace widths and distances (The etching quality of the copper traces was suboptimal.) and the neglecting interactions of different PCB board layers.

At both frequencies with minimal reflection, a far-field radiation pattern was calculated. For clarity, only the 24.518 GHz pattern is shown in Figure 2.5. The main lobe width was measured at approximately 16 degrees (Figure 2.6) along the 180-degree norm (for orientation refer to the red cone on the PCB visualization), with a peak gain of 18.6 dBi and side-lobe suppression of -13 dB. For the 90-degree norm, the main lobe width (Figure 2.7) was 30 degrees, with side-lobe suppression of -10 dB.

It's clear that especially in the 90-degree norm the accuracy isn't very good. Thus this header is not suitable for applications where accurate elevation information is needed. However if only low degree of accuracy is required or where wider field of view would be beneficial, this header is a good choice. Especially as the radar can gather more information about the environment faster than one

with narrower beam. In industry some surveillance radars are also designed to have a wide beam in elevation. Such as some Air Traffic Control radars [19] or even some ground radars [20]. Still usually solutions such as low/high beam switching are preferred instead of relying purely on a wide beam [21].

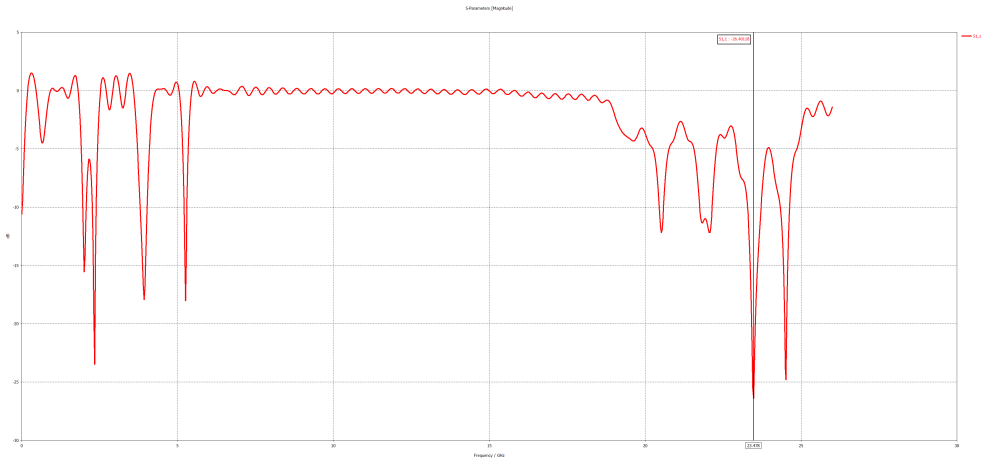


Figure 2.4: s_{11} parameter of the 24 GHz header

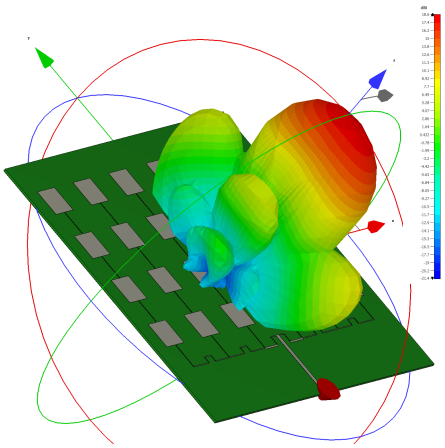


Figure 2.5: Radiation pattern of 24 GHz header – 3D view

2.3 122 GHz Header

The 122 GHz header is based on the TRX-120-001 transceiver, which, in addition to the essential components required for RF transmission and reception, also incorporates two on-chip antennas. It's designed to operate in the 122-123 GHz band, with output power ranging from -7 dBm to 1 dBm [23]. The chip is capable of detecting large targets at distances of up to 40 m [7].

Out of the box performance of the system is quite bad (Figure 2.8) with width of the main lobe being roughly ±40° in both E-plane and H-plane [23]. However, this can be significantly improved using the supplied collimator lens, reducing the main lobe width to ±4° [22] (see Figure 2.8).

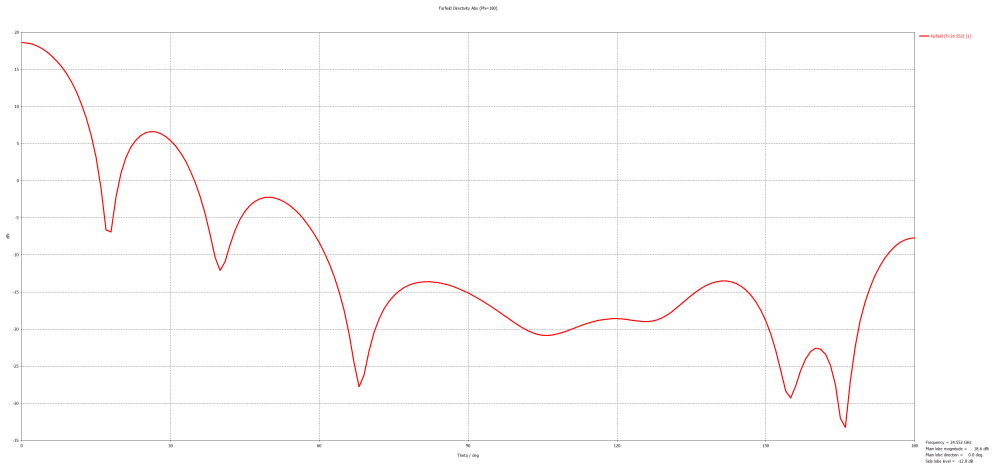


Figure 2.6: Radiation pattern of 24 GHz header – 180° norm

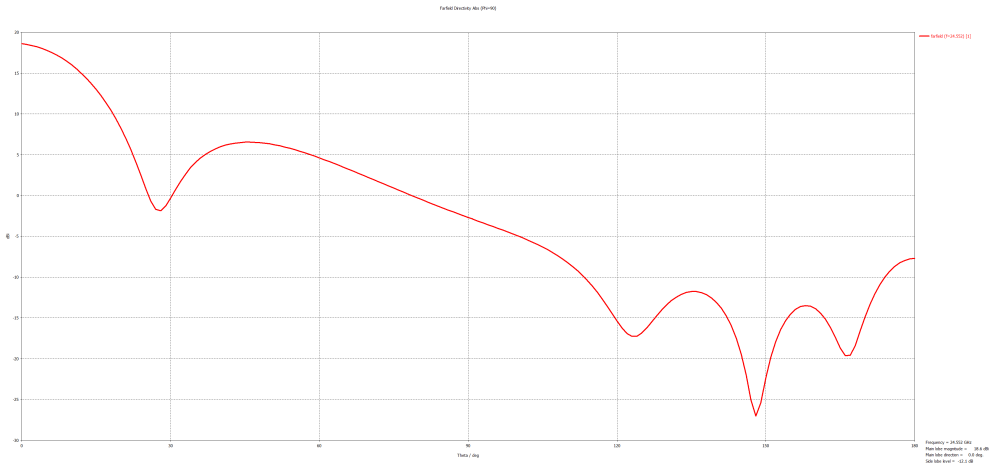


Figure 2.7: Radiation pattern of 24 GHz header – 90° norm

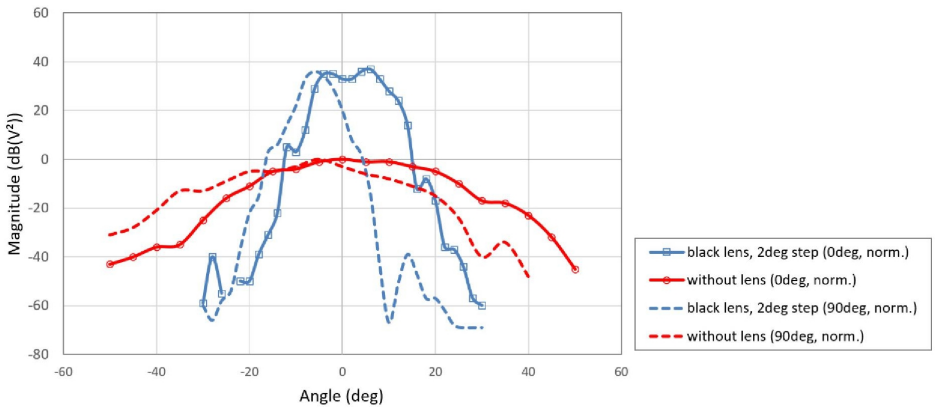


Figure 2.8: Radiation pattern of 122 GHz header comparison [22]

3. Rotary Platform

Following chapter outlines design process and the operation of a rotary platform specifically designed for SiRad Easy© radar system.

3.1 Platform Construction

As the platform needs to transmit data from the rotating section to the stationary base, a slip ring is required. Due to the relatively low transmission speed of the radar and the absence of special requirements such as waterproofing, an affordable model, UH3899-01-0810 from Senring, was selected. It is a classical contact slip ring that features a dedicated USB 2.0 connection along with 8 additional signal wires, with an advertised insertion loss of less than 2 dB [24]. More problematic than loss is however cross talk between signal wires when pitch control stepper motor is running. This necessitated an addition of small capacitor to the endstop signal wire to filter out high frequency noise.

While USB 2.0 connection doesn't seem affected by the crosstalk it's connection still isn't ideal. Without signal conditioner connected directly to the slipring output signal quality is bad enough that direct connection to PC is needed – thus radar needs to be some 20 cm from the PC. And even then depending on PC's USB port wiring some ports might exhibit issues. In addition the manufacturer opted for a non-standard male-male USB 2.0 connection, requiring a female-female adapter to connect the radar to the slip ring.

Due to the relatively low angular resolution of the radar, high platform precision was not a requirement. A basic 200-step stepper motor with a step size of 1.8° was deemed sufficient. In case user would require more granular control over the position it can be facilitated by microstepping.

Similarly high-speed movement is unnecessary for this application. As was already outlined the radar is capable at best of some 50 Hz update rate. With such update frequency even at low 8 RPM we travel a degree under 21 ms – any waster and we wouldn't get at least one update for each degree. At these RPMs standard 40 mm NEMA17 stepper motors are powerful enough to turn the rather stiff splipring.

The rest of the design is relatively simple. Fixed section mounts the slipring with a stepper motor positioned underneath, directly driving a shaft connected to a rotating base. The connection is secured using long M4 set screws that pass through the base, through the slip ring and hold the shaft in place. A 3D-printed housing serves only as a centering guide and is not under any load from shaft connection.

The rotating section features a simple A-frame design that elevates the radar, which is mounted on two ball bearings, allowing it to tilt freely. To control the pitch, a second stepper motor is mounted on the rotating platform and linked to the radar via a 2:1 down-gearing ratio using a standard 8 mm belt. An optical endstop, used for homing of the platform, is attached on the second support strut.

Unfortunately while the design offers some ability to tension the belt in currently printed configuration the adjustments range is not sufficient to fully tension the belt. While slippage is not a problem there is some slack in radar's tilt. However as the radar's capabilities in pitch direction aren't particularly high it was

decided not to reprint the whole assembly, only to fix issue in supplied 3D models.

Given that mechanical stresses are minimal, most parts can be 3D-printed using standard PLA filament. The only non-3D-printed components are the screws, bearings, and stepper motors. The final assembly (Figure 3.1) measures approximately 36 cm in height and has a footprint of roughly 20 × 20 cm.

(a) 3D render (b) Photo

Figure 3.1: Form of the final assembly

3.1.1 Platform Electronics

Electronic side of the platform is relatively simple. Only two main problems needed to be tackled – driving the stepper motors and implementing a homing mechanism.

Given the low load on the stepper motors and the platform's inability to accumulate significant momentum, a basic stepper driver without feedback control was deemed sufficient. For this purpose the A4988 stepper driver was chosen due to its low cost, microstepping capabilities and basic current control [25]. A minor drawback is the lack of any feedback from the driver to the microcontroller, including stall detection. To simplify the design, A4988 development kits were used and soldered onto a prototyping board, eliminating the need for a custom PCB. In addition if the need arieses A4988 development kit can be replaced by one with a different IC, such as TCM2209, as manufacturers keep the same pinout and dimensions on these. This might be necessary in application where platform loudness would be a serious problem as A4988 produces rather noisy driving signal. More modern drivers exhibit even 20 dB lower noise levels [26].

For homing implementation, two potential solutions were considered: Hall effect sensors and optical gates. Hall effect sensors offer the advantage of angle sensing, allowing correction for positional drift during operation; however, they require precise alignment. If the orthogonal Hall effect sensor is not perfectly placed along the axis of rotation, calibration becomes necessary [27]. While feedback would be beneficial, the microcontroller already tracks each step taken by the motor during normal operation, making it possible to determine the platform's position purely in software. Thus for simplicity and ease of integration, optical gates were selected.

The system is controlled by an ESP32 microcontroller. The ESP32-C6 version was chosen due to the author's extensive experience with this particular model. Still since the system does not require specialized peripherals or high processing power most other microcontrollers could have been use instead.

3.2 Platform Software

Given its widespread adoption as an industry standard for controlling similar machines, G-code over serial is a natural choice for the platform's communication format. Beyond the basic functionality typically offered by G-code interpreters, the platform must support additional features to reduce the user's manual control burden. These features include common features like defining movement limits, absolute/relative positioning mode, spindle mode or enabling automatic homing. Lastly the platform should support a programming interface that will allow application to send whole choreography of movements to the platform at ones and from that point only monitor its status.

For uplink communication, the platform must provide real-time information about its current position and speed. This data enables the user to make any mathematical corrections and properly interpret radar's gathered data.

To maximize efficiency in processing commands and ensure accurate stepper motor control, the program workflow is divided into three distinct layers, as illustrated by figure 3.2. The commonly used two-component architecture – where one component handles communication/command parsing and the other manages execution was deemed unsuitable for this use case. Such approach would complicate integration of programming interface and require just-in-time processing of commands, which could lead to performance issues.

In the chosen architecture, the degree of abstraction decreases with each successive layer, simplifying processing at each step. This design allows the final layer to operate with maximum efficiency, where transition from one command to next is next to instantaneous.

It is necessary to note that the platform doesn't store any permanent configuration, including microstepping settings This approach was taken as platform doesn't offer any easy way to show current setting and is expected to be only operated from computer application. Thus instead of needing to keep track of what settings are on PC, what are stored on the platform simpler approach was taken.

3.2.1 Communication Layer

The communication layer processes incoming data over the serial line. Whose efficient handling is facilitated with the aid of RTOS queues. Upon receiving command, string is parsed and either pushed to a queue to be executed or added to a programm declaration, in case user is currently writing a program. Some special commands are executed immediately, like those enabling emergency shut off. For complete overview of supported commands consult Appendix A.

Immediately after the parsing, a response is send to the user confirming whether the command was parsed correctly or not. However, as the communication layer does not a can not check command within context of all previous

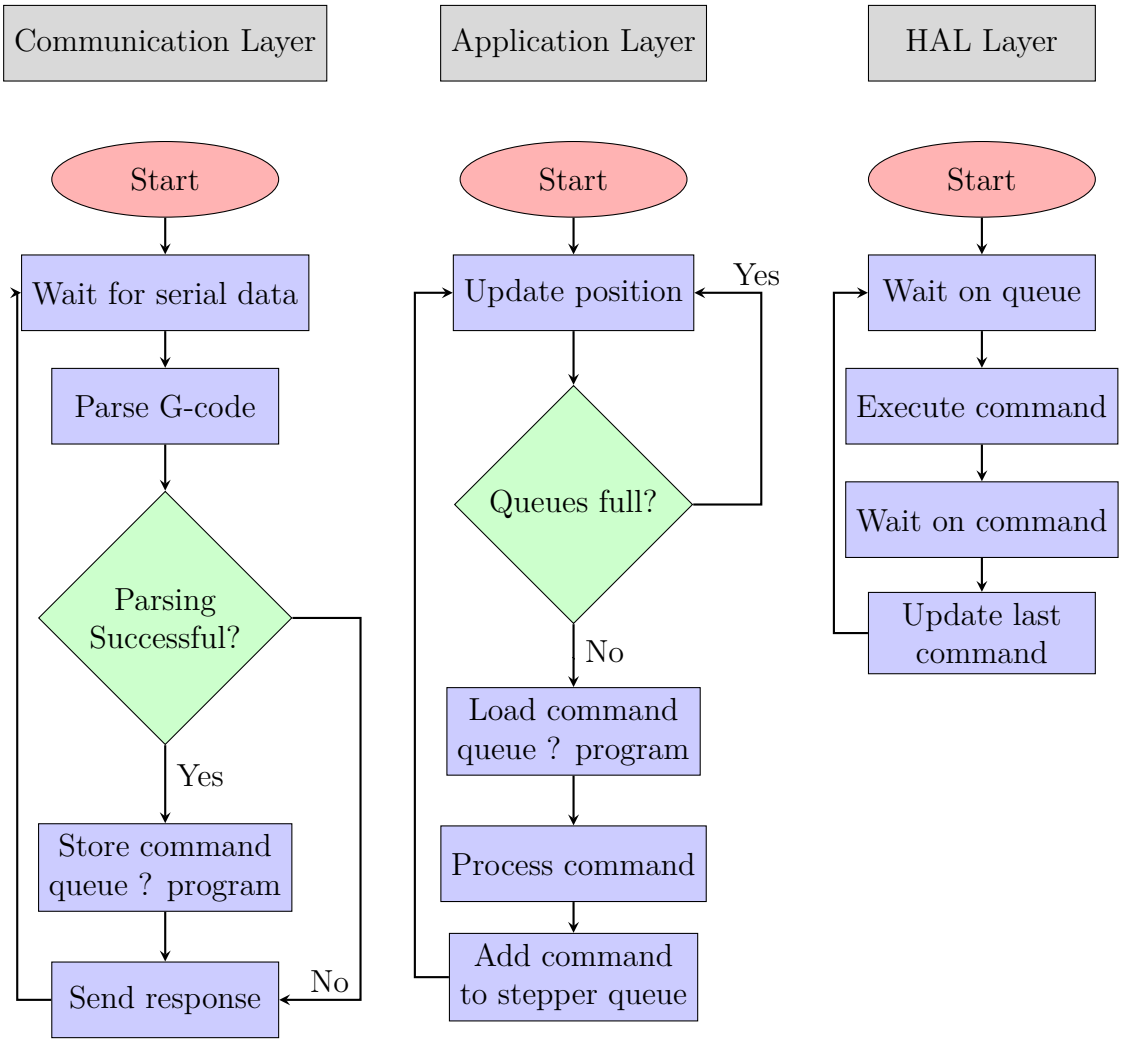


Figure 3.2: Program flow diagram

commands, it is possible that command will be parsed correctly but its execution will fail in the application layer.

3.2.2 Application layer

The application layer performs two primary functions: tracking current device position and scheduling commands to be sent to stepper motors. Aside from current position the program also keeps track of the end position of the last scheduled command. Thanks to this the application layer is able to make all necessary calculations to facilitate absolute positioning or enforce movement limits.

Key departure from standard G-code interpreters, like [28], is how the platform handles single-axis move commands. When a move command targets only one axis, the other axis remains free to read a next command and begin its execution. If this behavior is undesirable, the user must issue commands for both axes simultaneously. In relative positioning mode, a zero value results in no motion; in absolute positioning mode, the command must specify the current position to prevent movement.

This behavior is a necessary side effect of the spindle regime, which typically cannot be toggled on or off dynamically. Another consequence of spindle mode

is the requirement for separate positioning modes for each axis. Continuous rotation prevents calculating a move's end position, making it impossible to make calculation for absolute positioning commands – thus necessitating relative positioning. However it would be rather restrictive to force user to relative positioning on second axis, therefore the independent positioning modes.

In order to support other possible applications a manual override mode was also implementing. This enables the user to manually push a move command directly to stepper queues totally skipping the application layer. Primary usecase of this mode is to allow tracking of targets or usee in other application that require real time control over the platform. However in this regime no limits are enforced and the platform operates strictly in relative positioning mode.

3.2.3 HAL Layer

The final layer manages stepper motor control and provides the application layer with essential data for position calculations. In its loop, the program waits for the next command in the stepper queue. Upon receiving a command, it sets up execution, waits for one or both steppers to complete their movement, and then proceeds to the next command. Since limit and absolute positioning and limits calculations are handled in the application layer whole routine remains highly efficient – layer just moves given number of steps.

The main challenge was a generation of precise PWM signals (Used to control stepper motors drivers.) and stopping the signal generation after a specific number of steps. As generating PWM signal in software is unpractical dedicated EPS32C6 peripherals needed to be leveraged. There are two main options for this task: Remote Controlled Transceiver (RMT) and Motor Control Pulse Width Modulation (MCPWM) combined with Pulse Counter (PCNT). While RMT allows smooth PWM frequency adjustments, it has several drawbacks. Such as the fact that generating a specific number of pulses is supported only on newer ESP32 models [29], synchronization is restricted to its proprietary API and there is no straightforward way to track progress during a move [30].

For these reasons, MCPWM and PCNT were chosen. MCPWM handles pulse generation, while PCNT counts steps, enabling easy control over the movement and providing robust API for step count tracking [31]. The only limitation is the PCNT's 15-bit counter, which caps the maximum steps per move at $\pm$32,767.

Performance of the HAL Layer

Table 3.1 illustrates the stability of PWM generation by the MCPWM module at various speeds. Measurements were conducted using a Saleae Logic Pro 16 logic analyzer, with no microstepping enabled.

The results show that frequency deviation is minimal, though the generated speed is consistently marginally faster than the target, and the error increases slightly with higher speeds. Nevertheless, when measuring time of 24,000 steps at 120 RPM, the relative error in time duration (or speed) was only $\epsilon = -0.004\%$, demonstrating excellent accuracy.

An attempt was made to also measure the delay between switching commands, displayed in figure 3.3. The results indicate that the delay between commands is

Table 3.1: Stability of PWM generation

RPM	f_{desired} (Hz)	f_{low} (Hz)	f_{high} (Hz)	f_{avg} (Hz)
10	33.334	33.334	33.334	33.334
30	100	100	100.003	100.002
60	200	200	200.01	200.004
120	400	400	400.02	400.007

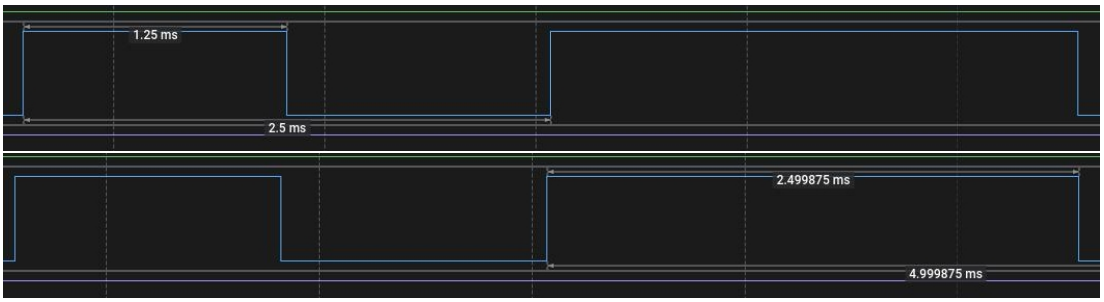


Figure 3.3: Moment of change between commands (120RPM ⇒ 60RPM)

imperceptible. Similar outcomes were also observed for other command combinations.

This demonstrates the efficiency of the HAL layer in managing stepper motor control and transitioning seamlessly between commands. As long as stepper queues are supplied with commands in advance, the platform can operate without noticeable interruptions.

4. Control Application

As previously stated control application for the whole system is written in MATLAB. There are two main distinct parts - one is managing the platform and the other is processing radar data. Alongside these there are some shared components, such as preferences menu.

4.1 Main Window

Upon starting the application user is presented with a main window (Figure 4.1). This window is primarily used to display radar data and control basic functionality of the application. On the sidebar user can enable serial connections to the radar and the platform, pause processing, save current visualization to jpg and stop the platform in case of an emergency. Lastly topbar is used to access preferences and platform control interface.

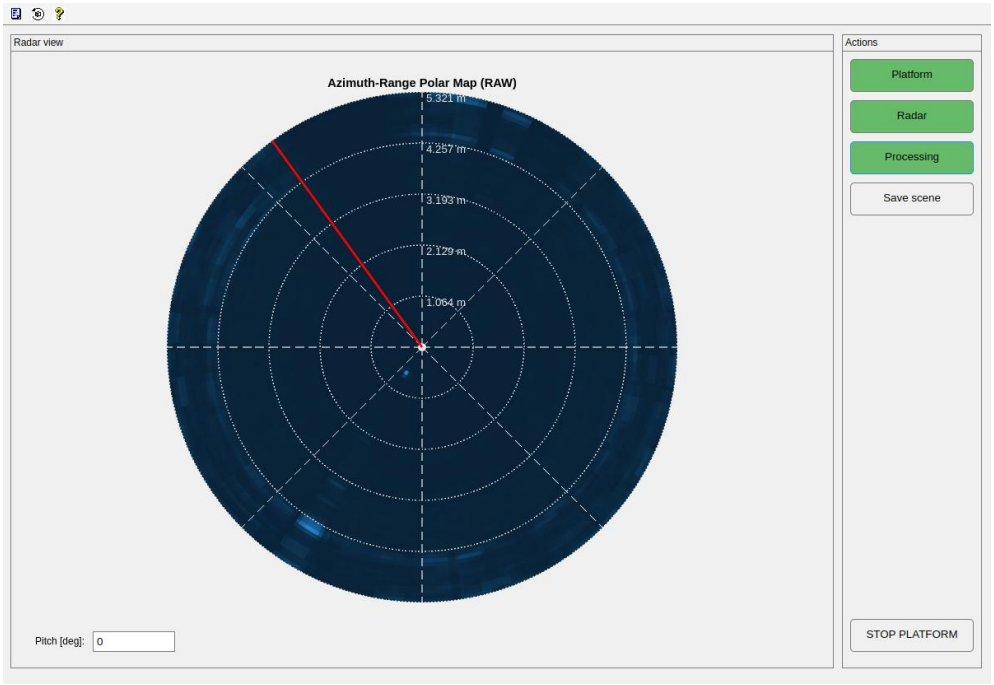


Figure 4.1: Main application window

4.1.1 Application Configuration

As this thesis concerns itself with rather generic implementation of surveillance radar system the application is designed to be as flexible as possible. For this purpose GUI menu was designed to enable configuration of all important elements - configuring the platform, radar, data processing and data visualization. However it is necessary to point out that given extensive configuration options user is expected to have some knowledge about the subject to achieve satisfactory results. Firstly – configuration options aren't checked against each other to verify if they aren't mutually exclusive. For example for speed calculation ramp averaging must be turned off, but as things stand application will allow user to continue even with

it on. Secondly – configuration of components such a CFAR or DBSCAN strongly depended on bandwidth, samples, FFT point count and other parameters. CFAR guard parameter of 3 might be too small on 5 GHz bandwidth with width of single range bin of 50 mm, but if with $BW = 500$ MHz it could be way too large. To aid with choosing correct parameters the application provides some basic information like dimensions of the bins; however it is up to the user to make the final decision.

Figure 4.2 shows the preferences menu, as one can see there are four main sections. First configures serial port connections to the radar and the platform. Second section handles configuration of the platform, followed by a section designated for radar configuration. Last section configures how the data are processed and via which method they are visualized.

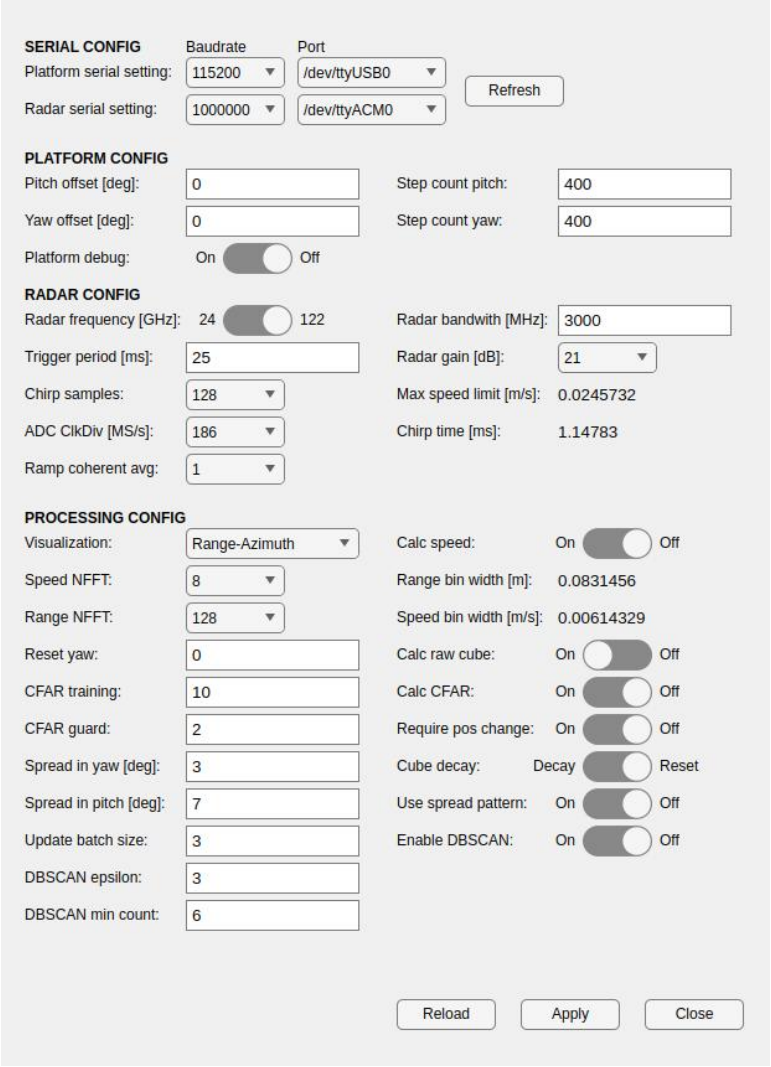


Figure 4.2: Preferences menu

User's preferences are stored in a INI file format in directory whose path depends on used operating system – `%APPDATA%/Local/fmcw/fmcw.conf` on Windows and `$HOME/.config/fmcw/fmcw.conf` for unix based systems. As INI format is easily human readable it is possible to edit the file directly.

4.2 Platform Control Interface

It is necessary to emphasizes that the platform control is entirely independent of the radar operation. User loads up program to the platform, starts its execution. From then application is only processing current position reports comming from the platform. There is no feedback from the radar to the platform based on processed data, although as the platform enables direct access to the stepper queues it would be possible to implement such a feature. Nor does processing take into account from which movements is a platform's current program comprised.

In preferences menu user can configure static offsets to the platform with the `Pitch offset` and `Yaw offset` options. This is useful in case zero position of the platform is not aligned with what user want's to be zero in data. Aside form offset step count configuration is also available here with `Step count pitch` and `Step count yaw`. Step count takes into accoun both microstepping and physical down gearing ratio. It is important to note that as the platform doesn't store step count in non volatile memory this setting is essential to maintain propper operation of the platform. The step count command is sent to the platform each time it is connected or config is changed. Lastly `Platform debug` toggle enables full debug output to be visible in platform control window.

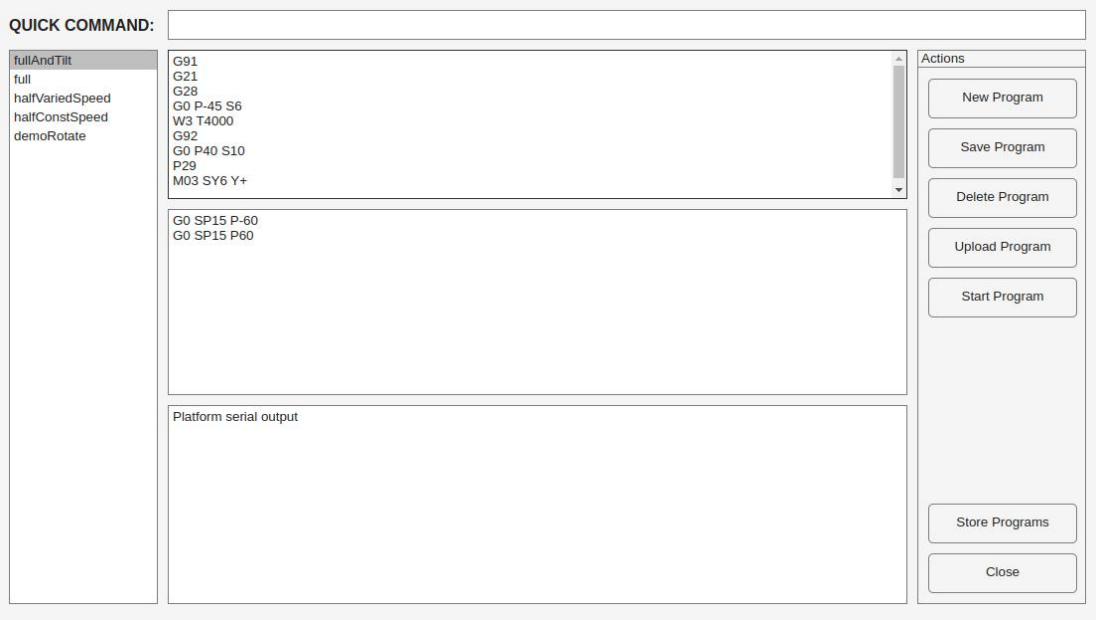


Figure 4.3: GUI of the platform control

Managing platform programs is done via a separate window (Figure 4.3). This interface enables user to load, edit and send programs to the platform. Left sidebar allows user to load stored programs by clicking on their name. Middle text boxes are used to edit programs header (top part) and program body (middle part) with bottom text field being used to display platforms debug output. On the right side are then buttons that handle basic operations over program – load, save, store to file, upload and start.

Commands are written one per line, their complete syntax is described in Appendix A. Control commands used to denote start of program, end of header and so on are automatically sent to the device when rest of the program is uploaded. To validate whether program was parsed correctly user can consult the

text window at the bottom of the screen which forwards platforms debug output to the user. In case user clicks the store button the program is stored in same INI file as preferences.

Backend side of the platform control is rather straightforward. After connection is established MATLAB serial interface calls a callback each time terminator sequence is found in received data. Received message is then parsed with debug output being forwarded to the user and positions logged with timestamp in a circular buffer. Radar data processing part can then request a list of positions for given time interval.

Platform control can also emit an event when platform finds itself close to a predetermined position. This location is specified in option `Reset yaw` with tolerance zone fixed at two degress on either side. In order to prevent multiple triggering of the event the platform will only emit it once per 1 s. This event allows reset of stored data after cycle is completed when `Cube decay` option is set to reset.

4.3 Radar configuration

Given that the radar's output is displayed in the main window there isn't any specialized interface dedicated just to the radar. Still in preferences menu there are some paramters that can be configured.

`Radar frequency` toggle chooses between 24 GHz and 122 GHz header. However this only affect configuration sent to the radar, it doesn't reflash the firmware what is also necessary to do. After base frequency `Radar bandwidth` can also be configred, both positive and negative values are accepted however manual verification is required to verify that the radar is working as expected. While manufacturer states a 3 GHz limit on the 24 GHz header and 5 GHz on the 122 GHz header [12], in practice, especially with the 24 GHz header, achievable bandwidth was usually much lower and its limit rather inconsistent.

User can also configure number of samples per chirp via `Chirp samples` and ADC rate with `ADC ClkDiv`. These two paramters determine time of a single ramp which is calcualted with equation (2.1). Together with gain, ramp time plays a crutial role in maintaining good signal-to-noise ratio.

Drop down menu `Ramp coherent avg` allows user to pick how many ramps will be averaged together. However use of this feature is generally not recommended unless surrounding environment is too cluttered (such as in a lab). Each ramps adds some 2 ms overhead to the radar's operation and makes speed calculation impossible.

Last paramters is `Trigger period` which dictates how often will the application send a trigger command to the radar. As previously stated the radar is at best capable of around 50 Hz update rate or 20 ms per chirp.

5. Radar Data Processing

Firstly it is necessary to outline what inputs and outputs of the whole processing pipeline are. Data to the application comes from two sources - serial connections to the radar and to the platform. These two streams are then paired together based on timestamps logged when data is received. Processing then does user chosen calculations – parts such as CFAR calculation, speed calculation and so on can be turned on or off. After that output data are stored in a radar cube, common method to manage radar's data that enables easy integration of more complex algorithms in the future [10].

Given that both CFAR and raw data (That is individual range-Doppler maps.) calculation might enabled at the same time application has two radar cubes. One is a 4D structure with dimensions of fast time × slow time × yaw × pitch, used to store complete range-Doppler map for each spatial degree. In this thesis this structure is only used for visualizing of raw data for given fixed pitch angle or showing range-Doppler map for given yaw and pitch. Any 3D visualization style of the raw data would be too cluttered and hard to interpret. Second cube is a 3D structure with dimensions of range × yaw × pitch, this one is used to store CFAR output.

In each case yaw resolution is fixed 1 degree thus the cube is 360° wide. Given quality of radar mounting bracket, 1.8° resolution of stepper and tighthness of radiation pattern 1° degree was deemed sufficient. As for pitch same resolution was kept with angles ranging form -20° to +80° being covered. Lower bound is limited by the radar's mounting bracket and upper bound is more or less arbitrary, slightly motivated by trying to perevent radar from seeing behind itself.

Main complicating factor in the processing was a need to keep the main thread responsive. Most time critical thing main thread is responsible for is timely readout of data from serial ports. If data can't be read right after its arrival assigned timestamps will be wrong and therefore data will be attributed to wrong position. In addition GUI starts to be quite unresponsive if resources of the main thread are stretched too thin.

To solve this problem processing heavily leverages MATLAB's Parallel Computing Toolbox, offloading as much as possible to different cores. Thorough whole processing pipeline there are mechanism of buffering data into larger batches and if needed discarding some in case software is unable to keep up. Unfortunately as MATLAB's parallelization capabilities with threads are rather limited – external language interface don't work, neither do certain file/memory mapping functions [32]. Thus the parallelization needed to be done with processes. This restrict the application to machines with at least 16 GB of RAM as each process (of which there are three) requires around 2 GB of RAM.

Procesisng pipeline can be divided into multiple distinct steps as follow:

1. Data acquisition: Data are read, simple FFT is calculated and sectrums are buffered into a circular buffer.
2. Data processing: Spectrums are retrived from the circular buffer, paired with positions and processed.
3. Cube update: Processed data are buffered into a ping-pong buffer. In case buffer is full the cube update routine is launched.
4. Visualization: After cubes are updated data are visualized.

5.1 Data Acquisition

Basic workflow of this step is showned on figure 5.1. Action starts as soon as MATLAB detects line terminator within incoming radar data stream. After whole frame from the radar is read, data are timestamped and I and Q channels buffered. Observer `dataProcessor.m` is then notified and left to retrieves the data when main thread is free.

After retrieval inphase and quadrature channels are merged to form an complex singal which is multiplied by a Hann window to prevent spectral leakage. Basic 1D FFT is then calculated with number of points set by `Range NFFT` paramter in preferences. In case fft order is larger than the number of samples zero padding is added. Resulting spectrum is again buffered, as spectrums will be subjected to futher processing they are kept in original complex form. Following this step program has in memory multiple FFT spectrums – these can be leveraged to enable range-Doppler map computation.

After spectrum is stored program verifies if the platform has moved. This check enables to save processing resources by not computing R-D map that would be immediately rewritten. After movement code than takes all spectrums (Except the last one when the platform has changed position.) and launches another processing step in a separate thread. In case user wishes for procesisng to always take place (e.g. they are verifying radar settings when platform is static) this position check can be skipped by toggeling `Require pos change`.

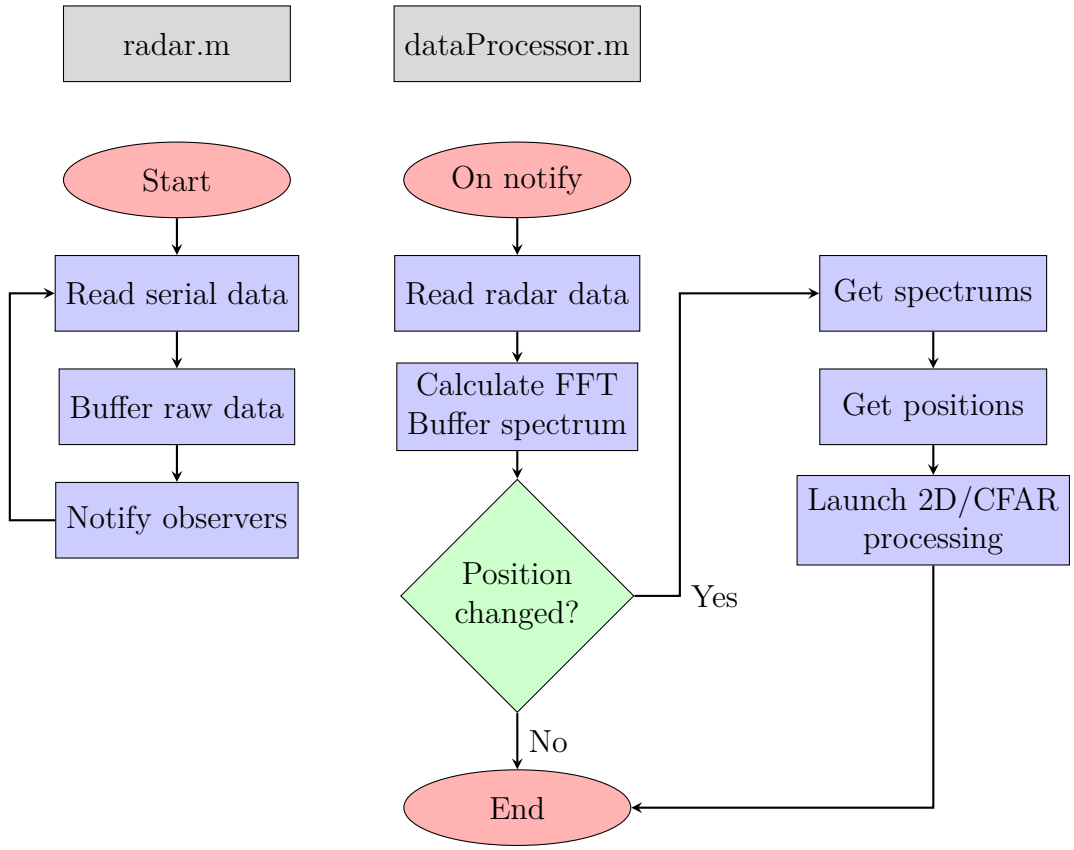


Figure 5.1: Processing flow - Data acquisition

5.2 Data Processing

This parallel process takes calculated FFT spectrums and their accompanying positions and processes them into single range-Doppler map and array with CFAR output. Basic outline of the processing flow is shown in figure 5.2.

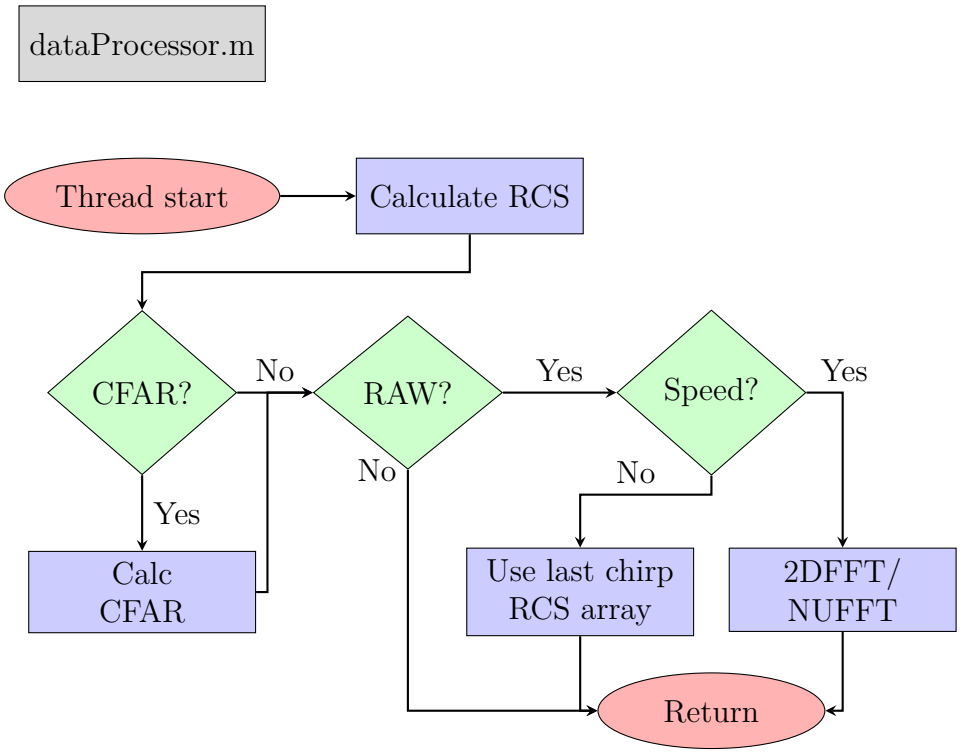


Figure 5.2: Processing flow - Data processing

In the first step last spectrum is converted to values proportional to the radar cross section of the target. Starting with equation (1.19) we can see that the target's RCS is proportional to the received power divided by fourth power of the distance to the target. Range can be calculated from manufacturer's supplied equation (2.2) and power is proportional to the square of values reported from the radar. Given that constants of the radar equation changed based on configuration calibration would be required in ordet to get real RCS values. However this calibration is not supported and thus onwards when refering to RCS we will be refering to the values proportional to the RCS and not real RCS of the object.

After RCS conversion if `Calc CFAR` is enabled CFAR algorithm will be executed on the latest chirp in the batch. In this application simple 1D CA-CFAR from MATLAB's Phased Array System Toolbox is used which parametrized with two variables – number of guard bins (Those are excluded from the average and are right next to bin we are testing.) and number of training bins (Those are used to calculate the average.) [33]. Both these paramters are configurable from preferences menu with `CFAR training` and `CFAR guard`.

If raw data calculation is enabled (Using `Calc raw cube` toggle.) the program takes one of two paths depending on whether speed calculation is toggled on or off with `Calc speed`. In case speed calculation is disabled the program just returns half of the original spectrum converted to RCS.

When speed calculation is enabled program firstly cuts of samples that are too far away from the position of the latest chirp. After that timing of chirps

are analyzed to determin whether NUFFT need's to be used. Decision which transformation to use is based solely on the ratio of maximal deviation of the chirp intervals to the median of the chirp intervals – if this ratio is over 20 % NUFFT is used. For more accurate measurement more complex method should have been used but as the radar timing data are not very precise not to mention radar can track only very slow moving objects, this method was deemed sufficient. For simmilar resasons no correction for the platform's movement is applied. Output range-Doppler map is then shifted in order to get 2D array with `Range NFFT` × `Speed NFFT` number of elements with speed going from $-\frac{\text{Speed NFFT}}{2} \cdot w$ to $\left(\frac{\text{Speed NFFT}}{2} - 1\right) \cdot w$, where w is speed bin widht calcualted with (1.18).

5.3 Cube Update

Last step of processing pipeline handles updates of radar cubes (Figure 5.3). Range-Doppler maps and CFAR data are firstly buffered in a ping-pong buffer – where one is being used to update the cube while the other is being filled with new data. Upon the buffer is filled they switch roles and cube update routine is launched.

As matlab processes do not share same memory space by default, in order not to have redundant copies of cube matlab's memory mapping is used. Using function `memmapfile` the cube is mapped to a file on the disk which can be shared between two processes [34]. This approach might lead to some performance degradation if the cube is too large and cannot be held in RAM but in testing it was found that for cubes of size 512MB (roughly cube for range-Doppler map of 256 × 16) the program was still able to keep up with the radar.

Update routine for cubes containing CFAR or raw data, are quite simmilar, in the case if spread pattern is disabled. New data are just added to the cube to their respective positions where they replace old ones. In case cube is set to decay (`Cube decay` option is set to decay.) whole cube is multiplied by a decay factor that is derived from platforms movement speed. The slower the platform moves the slower the data decay. Using this option leads to an output more visually resembling radar traditional analogue radar systems where most recent data were brighter than the older ones. However user needs to keep in mind that color changes doesn't mean a change in RCS at that position. In addition this operation is rather computationally expensive and is the primary reason updates to the cube must be buffered.

Even on smaller cubes of around 256 MB this operations takes in worse case some 50 ms to complete on a laptop i7-10210U CPU. If change of value would be undesirable behaviour decay can be turned off in the settings in favour of hard reset of the cube based on trigger on specified yaw position. Rest of the processing is then fast enough to keep up with the data stream without any buffering.

Enabling spread pattern leads to degradation of quality of the data in exchange for better visualization quality – especially in range-azimuth visualization style. Instead of adding new data just to one position in the cube the range-Doppler map is spread over a larger area with aid of a 2D matrix, made up of a 2D Gaussian function. As the data in series are usually close to each other in terms of position update process can be made more efficiently. Instead of directly adding these contributions to the full cube much smaller cube is created. That cubes

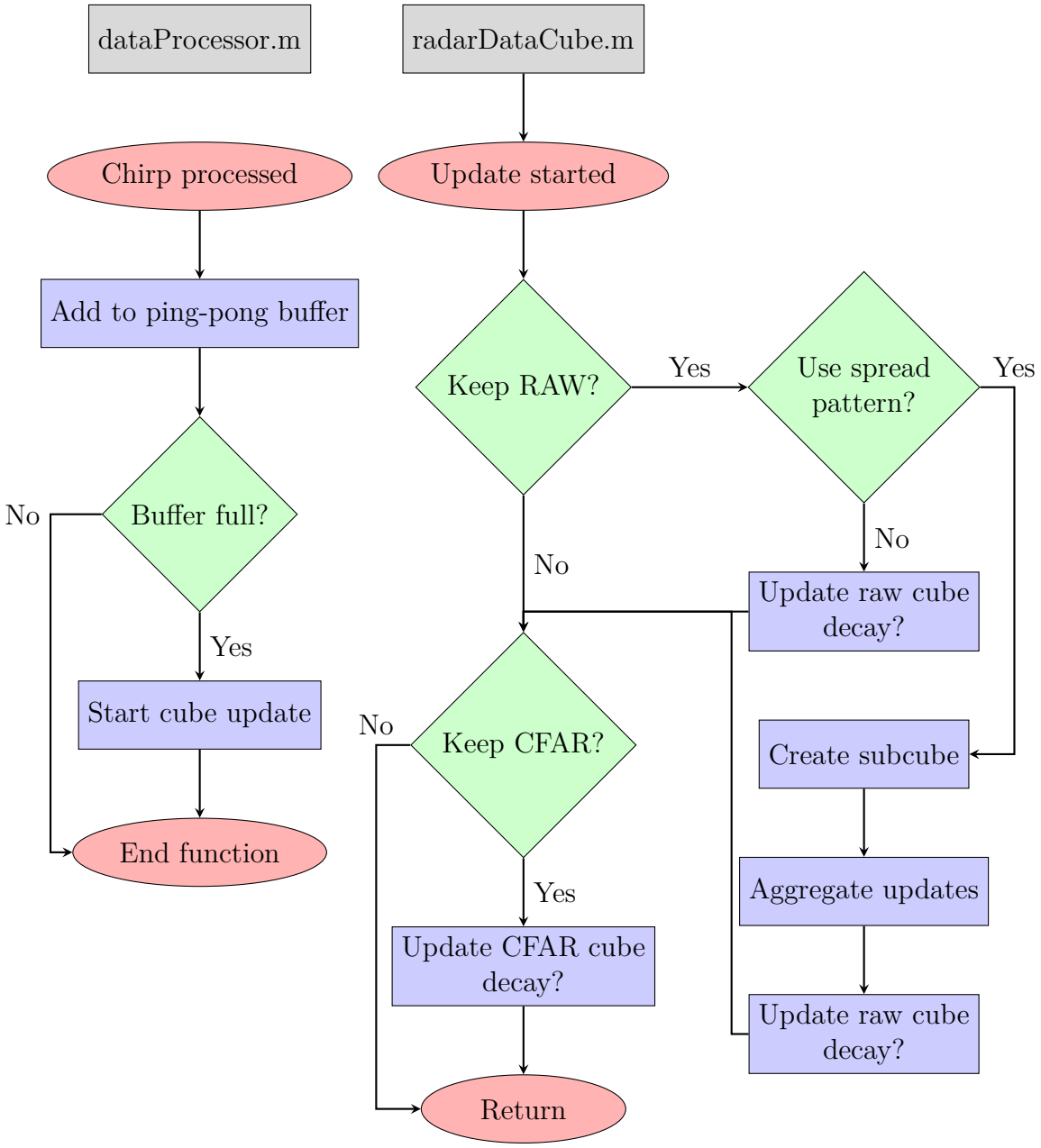


Figure 5.3: Processing flow - Cube update

covers only the area that will be updated with the new data and is later added to the full cube in one go. This enables significant speedup as smaller memory size enables better cacheing. Still in case of a more performant radar system this method would be unncessary not to mention has very little basis in radar data processing theory.

Due to lacking performance of MATLAB on author's machine some parts of the cube update routine were implemented in C++ with the aid of MATLAB's MEX interface. This enabled the use of AVX2 x86 instruction set extension to speed up floating point operations. Chiefly decaying of the cube exhibited roughly two fold performance increase thanks to SIMD instructions.

5.4 Visualization

After cube update finishes program returns to the main thread and the data are visualized. Only way user can see the data is via visualization. Text output or other methods are not available. However the application offers a number of different visualization styles to suit the user's requirements.

1. Range-Azimuth: Polar plot with range on the radius and azimuth on the angle.
2. Range-Doppler: Classical xy plot with range on x axis and Doppler on y axis, if speed is not calculated the y axis is replaced with RCS value.
3. Target-3D: CFAR data are displayed in 3D cartesian space

5.4.1 Range-Azimuth

This visualization style is used to display the data in polar plot for fixed pitch angle that is chosen by an text box. The demonstration of Range-Azimuth map can be seen on figure 4.1, which captures testing with 122 GHz header with rather large bandwidth of 3 GHz giving only some 5 meters of range. On the picture only raw data are displayed however this visualization style is able to also display CFAR data or overlay both of them.

Instead of calculating projection into horizontal plane the data are displayed in a conical projection. That is, no additional calculations are made and the cone we get by fixing the pitch angle is simply stretched to a circle. To aid with orientation in a plot simple line markers are added to provide range and angle reference points.

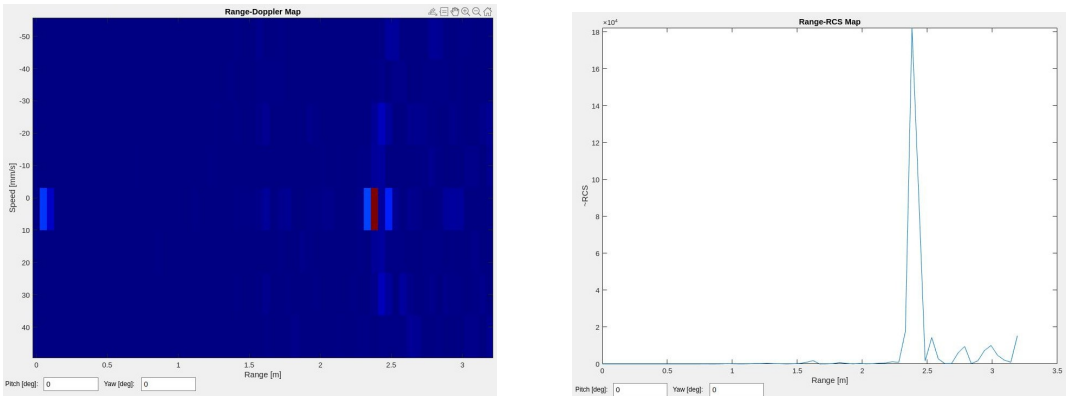
This visualization style might be especially usefull when paired with 24 GHz header. Due to it's rather wide radiation pattern the radar is able to detect objects in quite a large area. This can be even more extended if the spread pattern is enabled where more pitch angles will influcence the data. Thus for a realization of classical surveillance radar this method of visualization might be most versatile.

5.4.2 Range-Doppler

Picking this visualization style from the preferences menu will display the data in a classical x-y plot for a fixed yaw and pitch angle. Just as with Range-Azimuth map both angles are configurable via the textboxes at the bottom of the screen. In cases when speed calculation is disabled the y axis is replaced with RCS value.

Main purpose of this visualization style is to provide verification method for the radar data processing and setting. As the devkit doesn't report much of a useful errors and its technical documentation is sometimes incorrect (e.g. in allowed bandwidth values) it is useful to have a way to verify that the radar is working correctly.

On figure 5.4 we can see two different visualization styles both displaying the same static scene. As previously stated given rather poor sampling rate speed calculation is not very useful and thus no complex testing has been done of it.



(a) Range-Doppler map (b) Range-RCS

Figure 5.4: Range-Doppler and Range-RCS maps

5.4.3 Target-3D

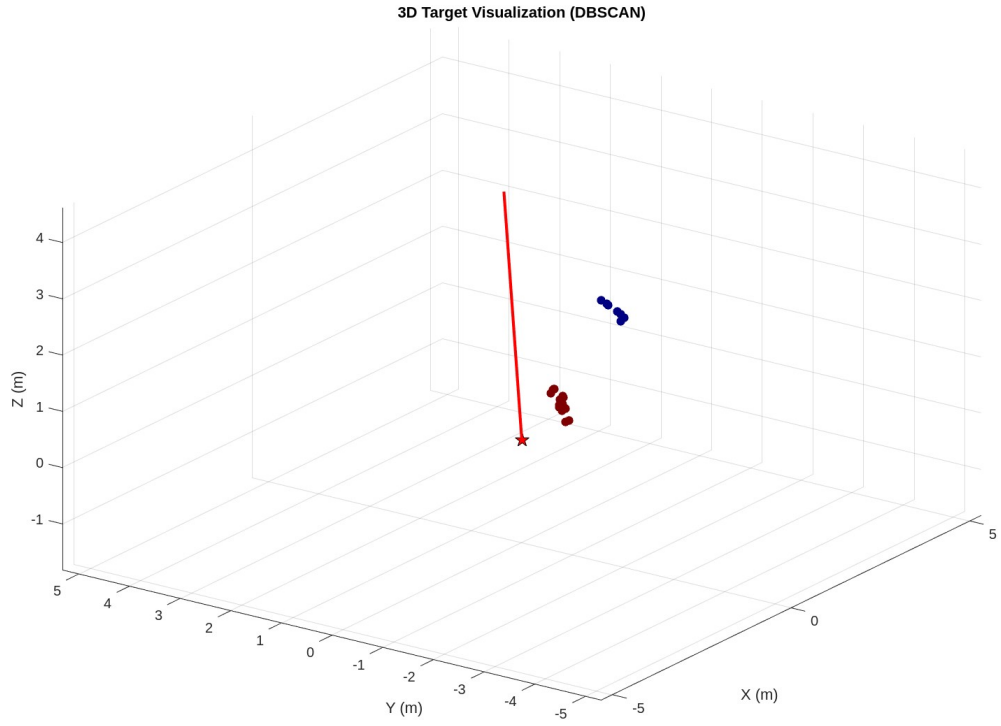


Figure 5.5: 3D Space visualization method

This visualization style is used to display CFAR data in 3D cartesian space (Figure 5.5). Visualization method is rather simple with just positions being recalculated from the range, azimuth and pitch angles to x, y and z coordinates using trigonometric functions. In case decay is enabled then small threshold is used to remove points whose values was sufficiently regraded. To aid with orientation within the plot curret heading of the platform is displayed as a red line.

If environment is too cluttered and CFAR generates too many points the user can enable basic clustering with MATLAB's built in DBSCAN function [35]. Gerneral idea of DBSCAN is that for a given point in the dataset distances to other points are calculated and compared with user defined threshold ε, in case

sufficient number N of points is found within the threshold the point is marked as part of an cluster [36]. In MATLAB's in algorithm distance threshold and minimal number of points are both configurable variables with other input being just the points list [35]. As original data are in polar coordinates system problem arises with calculating distances given that in such system distance between two points of given fixed yaw and pitch varies widely with range. Not to mention with increasing distance the density of data points decreases. Aside from that we also have significantly more points closer to the radar as space is much more densely scanned there compared to the far end of the range.

These problems can be overcome using many different methods like using adaptive thresholds in relation to radius [36] or relying on more clever axis systems then just polar or cartesian [37]. However as in this thesis as DBSCAN is more of an illustration of possible postprocessing no such complex methods were implemented. Both ε and N parameters are left to the user to configure with `DBSCAN epsilon` and `DBSCAN min count`. As for the distance metric points are at firstly converted to cartesian coordinates then distance is calculated using standard euclidean formula. This distance is then divided by averaging radiuses of points. Still it needs to be said that, while dividing by the mean of distances is sometimes used when working with spherical data [38], author is not aware of any literature that would suggest use in DBSCAN. Especially on longer distance this methods will lead to significant errors.

Conclusion

Goal of this thesis was to realize a surveillance radar system based on FMCW technology. This technology should enable accurate distance measurements of targets with a relatively low power consumption. Instead of more conventional MIMO systems a simpler solution was proposed with a single RX and TX antenna that relies on mechanical steering of the radar beam.

Using off-the-shelf components and 3D printed parts a custom rotary platform was designed and constructed. All be it minor issues in regards to the belt tension on pitch axes in enables controlling of the radar position in both yaw and pitch. Normally with 1.8° degree of accuracy that can be extended with the use of microstepping. Quality of life features such a automatic homing system or limits to the rotation were also implemented. Whole system is controlled by an ESP32C6 microcontroller which interprets G-code like commands and drives the stepper motors. Due to its similarities to other G-code base systems it should be readily adaptable to other systems. In addition the platform was designed to support capabilities that aren't strictly necessary for the purpose of this thesis. During tests platform was able to execute commands accurately and without noticeable delays.

Capabilities of SiRad Easy© evaluation board were analyzed and were found to be rather lacking for application in a surveillance radar. It's low and inconsistent reporting rate of 50 Hz limits the maximum detectable speed to tens of mm per second. This effectively eliminates any possibility of tracking moving targets which is a common use case of surveillance radars and severely limited possible scope of this thesis. Whole system is only suitable for essentially static scenes with binary detection if target is or is not present. However in these cases the radar is still able to provide a good resolution across wide variety of ranges given the ability to switch between 24 GHz and 122 GHz headers.

Control application for the surveillance radar was developed in MATLAB. It integrates both the rotary platform management and radar data processing into one combined package. The data processing pipeline is relatively standard relying on common techniques such as FFT, CFAR, DBSCAN and stores data in standard radar cubes. Heavy parallelization was employed to ensure that the processing is done in a timely manner. This enables application to maintain speed even if costly operations, entailing millions of floating point operations, such as a decaying of the whole cube are performed.

Given rather generic design requirements there is a large degree of customization possible to the processing pipeline. From simple things such as the number of FFT points, step count per rotation to more complex ones such as configuring CFAR parameters can all be tailored to a specific application. One major downside of this approach is that the user is required to have a good understanding of the underlying algorithms and principles.

Only visual outputs were implemented in this thesis with both 3D and 2D visualizations supported. Classical 2D Range-Azimuth map is probably the most useful and given the radar's radiation pattern provides information about sizable part of the environment. In case of 3D visualization only CFAR data are visualized in exchange for a more detailed view of the environment by providing pitch information. Were radar to be deployed in a more cluttered environment

DBSCAN algorithm can be employed to filter out unwanted noise. However it's implementation in the thesis is rather illustrative and not optimized for performance or accuracy.

In addition the whole processing pipeline is written in a way that, if radar module was exchanged for a faster one, capabilities could be extended while keeping most of the codebase similar. More complex operations such as 2D FFT or cube updates are executed in parallel processes and are triggered by platform movement, not for each radar update. Therefore increase in the number of chirps wouldn't have a significant impact on the performance.

Still in case where number of FFT points either in speed or range would be significantly larger that what has been tested (Cubes to size of around 500 MB were validated.) different approach to cube update would be required. First possibility would be leverages GPU acceleration to handle operations in the cube update. Or second in case of solely CPU the cube would need to be split into smaller chunks where only few ones would be loaded into RAM at one time – depending on the current position of the platform and direction of travel. However decay as it is implemented now would be close impossible to realize.

In conclusion basic surveillance radar system was successfully designed and implemented. It has ability to detect targets in a wide area with basic resolution of 1 degree in yaw/pitch and from centimeters to decimeters in range. In addition extensive customization of the processing pipeline allow user to fine tune the system to their needs. However due to limitations of the radar module it is not suitable for any kind of applications requiring tracking of moving targets.

Bibliography

[1] Sandeep, Rao. MIMO Radar [online]. 2018 [visited on 2025-05-14]. Available from: https://www.ti.com/lit/an/swra554a/swra554a.pdf.

[2] Nowok, Sandra; Kueppers, Simon; Cetinkaya, Harun; Schroeder, Martin; Herschel, Reinhold. Millimeter wave radar for high resolution 3D near field imaging for robotics and security scans. In: *2017 18th International Radar Symposium (IRS)* [online]. IEEE, 2017, pp. 1–10 [visited on 2024-11-02]. ISBN 978-3-7369-9343-3. Available from DOI: 10.23919/IRS.2017.8008132.

[3] Vivet, Damien; Checchin, Paul; Chapuis, Roland. Localization and Mapping Using Only a Rotating FMCW Radar Sensor. *Sensors* [online]. 2013, vol. 13, no. 4, pp. 4527–4552 [visited on 2024-11-01]. ISSN 1424-8220. Available from DOI: 10.3390/s130404527.

[4] Brooker, Graham. Long-Range Imaging Radar for Autonomous Navigation. 2005. Available also from: https://ses.library.usyd.edu.au/handle/2123/658.

[5] Jankiraman, Mohinder. *FMCW radar design*. Boston: Artech House, [2018]. ISBN 978-1-63081-567-7.

[6] Indie semiconductors GmbH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: User Guide* [online]. 2021. Version 2.5 [visited on 2024-11-19]. Available from: https://downloads.ffo.indiesemi.com/datasheets/User_Guide_Easy_Simple_V2.5.pdf.

[7] Indie semiconductors GmbH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: User Guide*. 2017. Version 2.1.

[8] *2-axis (XY) Rotation Stage Motorized XYZ Translation Stage* [online]. [visited on 2024-11-24]. Available from: https://www.standa.lt/products/catalog/custom_engineering?item=658.

[9] *Two Axis Rotary Indexing Tables* [online]. [visited on 2024-11-24]. Available from: https://www.carlhirschmann.us/en/Rotary-tables/Products/Two-Axis-Rotary-Indexing-Tables.

[10] Richards, Mark A. *Fundamentals of radar signal processing*. Third edition. New York: McGraw Hill, [2022]. ISBN 978-1-260-46871-7.

[11] *Plán spektra ČTU od 24 po 170 GHz* [online]. [visited on 2025-01-23]. Available from: https://spektrum.ctu.gov.cz/kmitocty?filter%5BfrequencyFrom%5D=24%5C&filter%5BfrequencyFromUnit%5D=GHz%5C&filter%5BfrequencyTo%5D=170%5C&filter%5BfrequencyToUnit%5D=GHz.

[12] Indie semiconductors GmbH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: System & Protocol Description* [online]. 2017. Version 2.1 [visited on 2025-01-25]. Available from: https://downloads.ffo.indiesemi.com/datasheets/Protocol_Description_Easy_Simple_V2.1.pdf.

[13] Adler, Viktor. *Frequency-Modulated Continuous-Wave (FMCW) Radar* [online]. 2023. [visited on 2025-05-15]. Available from: https://moodle.fel.cvut.cz/pluginfile.php/348825/mod_resource/content/3/FMCW_Radary.pdf.

[14] Suleymanov, Suleyman. Design and Implementation of an FMCW Radar Signal Processing Module for Automotive Applications. 2016. Available also from: http://essay.utwente.nl/70986/.

[15] Piper, S.O. Homodyne FMCW radar range resolution effects with sinusoidal nonlinearities in the frequency sweep. In: *Proceedings International Radar Conference*. 1995, pp. 563–567. Available from DOI: 10.1109/RADAR.1995.522609.

[16] Chaudhari, Qasim. *FMCW Radar Part 3 – Design Guidelines* [online]. [visited on 2025-05-03]. Available from: https://wirelesspi.com/fmcw-radar-part-3-design-guidelines/.

[17] Rohling, Hermann. Radar CFAR Thresholding in Clutter and Multiple Target Situations. *IEEE Transactions on Aerospace and Electronic Systems* [online]. 1983, vol. AES-19, no. 4, pp. 608–621 [visited on 2025-05-14]. ISSN 0018-9251. Available from DOI: 10.1109/TAES.1983.309350.

[18] Indie semiconductors GmbH. *TRX_024_007: Datasheet* [online]. 2019. Version 1.5 [visited on 2024-11-19]. Available from: https://siliconradar.com/datasheets/Datasheet_TRX_024_007_V1.5.pdf.

[19] *S-band Air Traffic Control Radar Antenna System* [online]. [visited on 2025-05-10]. Available from: https://www.kratosdefense.com/-/media/k/pdf/s/a/icao-s-band-atc-radar.pdf.

[20] *Revolution 360 Ground Surveillance Radar* [online]. [visited on 2025-05-10]. Available from: https://www.blighter.com/products/revolution-360-ground-surveillance-radar/.

[21] Wolff, Christian. *High and Low Beam* [online]. [visited on 2025-05-15]. Available from: https://www.radartutorial.eu/06.antennas/High-%5C%20and%5C%20Low-Beam.en.html.

[22] Indie semiconductors GmbH. *Collimator Lens* [online]. 2021. Version 1.1 [visited on 2025-01-23]. Available from: https://siliconradar.com/datasheets/Datasheet_Collimator_Lens_V1.1.pdf.

[23] Indie semiconductors GmbH. *TRX_120_001: Datasheet* [online]. 2022. Version 1.6 [visited on 2024-11-19]. Available from: https://siliconradar.com/datasheets/Datasheet_TRX_120_001_V1.6.pdf.

[24] *UH3899-01 Series USB2.0 Signal Slip Ring* [online]. [visited on 2024-11-10]. Available from: https://www.senring.com/usb-conductive-slip-ring/single-channel/uh3899-01.html.

[25] *A4988 Datasheet* [online]. [visited on 2024-11-23]. Available from: https://www.allegromicro.com/~/media/Files/Datasheets/A4988-Datasheet.ashx.

[26] Klements, Michael. *TMC2208 Vs A4988 Stepper Motor Driver Sound Level Comparison* [online]. [visited on 2025-04-30]. Available from: https://www.the-diy-life.com/tmc2208-vs-a4988-stepper-motor-driver-sound-level-comparison/.

[27] Chen, Xiao; He, Xiran; Zhang, Ying; Gao, Bo. Angle Measurement Method Based on Cross-Orthogonal Hall Sensor. In: *2023 IEEE 16th International Conference on Electronic Measurement & Instruments (ICEMI)* [online]. IEEE, 2023-8-9, pp. 153–158 [visited on 2024-11-23]. ISBN 979-8-3503-2714-4. Available from DOI: 10.1109/ICEMI59194.2023.10270587.

[28] *GCode dictionary* [online]. [visited on 2024-11-24]. Available from: https://docs.duet3d.com/en/User_manual/Reference/Gcodes.

[29] *RMT Based Stepper Motor Smooth Controller* [online]. [visited on 2024-10-03]. Available from: https://github.com/espressif/esp-idf/tree/master/examples/peripherals/rmt/stepper_motor.

[30] *Remote Control Transceiver (RMT)* [online]. [visited on 2024-10-03]. Available from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/rmt.html.

[31] *Pulse Counter (PCNT)* [online]. [visited on 2024-11-09]. Available from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/pcnt.html.

[32] *Run MATLAB Functions in Thread-Based Environment* [online]. [visited on 2025-05-12]. Available from: https://www.mathworks.com/help/matlab/matlab_prog/run-functions-on-threads.html.

[33] *Constant False Alarm Rate (CFAR) Detection* [online]. [visited on 2025-05-08]. Available from: https://www.mathworks.com/help/phased/ug/constant-false-alarm-rate-cfar-detection.html.

[34] *Share Memory Between Applications* [online]. [visited on 2025-05-08]. Available from: https://www.mathworks.com/help/matlab/import_export/share-memory-between-applications.html.

[35] *DBSCAN* [online]. [visited on 2025-05-12]. Available from: https://www.mathworks.com/help/stats/dbscan-clustering.html.

[36] Kellner, Dominik; Klappstein, Jens; Dietmayer, Klaus. Grid-based DBSCAN for clustering extended objects in radar data. In: *2012 IEEE Intelligent Vehicles Symposium* [online]. IEEE, 2012, pp. 365–370 [visited on 2025-05-12]. ISBN 978-1-4673-2118-1. Available from DOI: 10.1109/IVS.2012.6232167.

[37] Sun, Xiaoxiao; Sajda, Paul. Circular Clustering With Polar Coordinate Reconstruction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* [online]. 2024, vol. 21, no. 5, pp. 1591–1600 [visited on 2025-05-12]. ISSN 1545-5963. Available from DOI: 10.1109/TCBB.2024.3406341.

[38] Fisher, N. I.; Lewis, Toby; Embleton, Brian J. J. *Statistical analysis of spherical data*. Cambridge: Cambridge University Press, 1993. ISBN 0-521-45699-1.

List of Figures

List of Tables

Appendix

Appendix A – G-code glossary

Following sections acts as glossary of all G-code commands used in the platform control. Mode indepth description of the commands can be found in the source code.

Constants

- Parameters defined in Kconfig (require firmware recompilation):
- `STEPPER_Y_STEP_COUNT`, `STEPPER_P_STEP_COUNT`: Steps per rotation (modifiable via `M92`, non-persistent)
- `STEPPER_[Y/P]_PIN_DIR`, `STEPPER_[Y/P]_PIN_STEP`: GPIO pins for direction/step signals
- `STEPPER_[Y/P]_PIN_ENDSTOP`: Endstop detection pins
- `STEPPER_MAX_SPEED`: Maximum RPM (hardware limit)
- `STEPPER_DEFAULT_SPEED`: Default motion RPM
- `STEPPER_MIN_SPINDLE_TIME`: Minimum step interval of rotation in spindle mode
- `STEPPER_HAL_TIMER_PERIOD/RESOLUTION`: PWM timer period/resolution, affects achievable RPM range

Features

- **Units**: Switch between degrees and steps using `G20`/`G21` respectively
- **Positioning Modes**:
  - Absolute (`G90`): Automatically normalizes angles to [0, 360°] or [0, STEP_COUNT]
  - Relative (`G91`): Limited to ±32767 steps per command, or respective angle counterpart
  - in absolute positioning devices moves in the shortest path to the target
- **Spindle**: continous rotation mode
- **Limits**: Limits on rotation can be imposed on both axes
  - limits are applied only to one rotation – it is not possible to restrict to e.g. two full rotations in one direction a nd one in the other
  - `low < high`: Valid range [low, high]
  - `low > high`: Valid range [low, 360°] ∪ [0, high]
  - Out-of-range targets snap to nearest valid position
- **Programming**: G-code program can be upload to the device and executed
- **Synchronization**: If command is issued to both axes they will wait for each other, of not other axis will be free to execute next command

Motion Control Commands

Axes are denoted as `Y` for yaw and `P` for pitch. The `S` parameter is used for speed for both axes, while `SY` and `SP` are used for speed of yaw and pitch respectively. In case of spindle `Y` or `P` are not followed by step count/angle but instead rotation direction.

`M80` Enable high-voltage power (enables drivers)
`M81` Disable high-voltage power (disables drivers)
`M82` Emergency stop (clears command queues)
`G20` Set units to degrees
`G21` Set units to steps
`G90 [Y/P]` Set absolute positioning Mode

- absolute positioning is not available in spindle mode, if you issue a M03 command to stepper in absolute positioning mode, it will be automatically switched to relative positioning and throw an error
- if no argument is provided both axis are switched to relative positioning

`G91 [Y/P]` Set relative positioning Mode
`G28 [Y/P]` Auto-home specified axis(es)
`G92 [Y/P]` Set current position as zero
`G0 Y<val> P<val> [SY/SP<rpm>]` Stepper-mode movement
`M03 Y/P<+/-> [SY/SP<rpm>]` Start spindle mode (continuous rotation)
`M05 [Y/P]` Stop spindle mode
`M201 LY/HY/LP/HP<angle>` Set angular limits (degrees/steps)
`M202 [Y/P]` Disable limits

Special Commands

These commands bypass certain sections and safety checks of the firmware. Use with caution.

`G3 Y/P<steps> [SY/SP<rpm>]` Bypass scheduler and adds movement directly to the queue

- limits are not and cannot be checked
- absolute positioning is not available
- all values are interpreted as steps regardless of current unit setting
- *Warning*: Use `M82` before `G3` to maintain position integrity

`W3 T<ms>` Application-layer delay

- useful to wait for end of homing process

Programming Commands

`P90 <id>` Start programming mode (overwrites existing program if any)
`P91` Transition from header to main program body
`P92` Finalize program (discarded if loops unclosed)
`P21 I<iterations>` For-loop declaration
`P22` End loop block
`P29` Infinite loop marker (header only) - main body will be executed infinitely
`W0 Y/P<sec>` Wait in seconds
`W1 Y/P<ms>` Wait in milliseconds

Uplink Protocol

- `!P <timestamp>, <yaw>, <pitch>`: Position update (20ms interval)
- `!R OK`: Command acknowledgment
- `!R ERR <code>`: Command error

Code	Description
1	Malformed command syntax
2	Invalid arguments
3	Queue lock failure
4	Unsupported command
5	System busy (homing/program running)
6	Runtime exception
7	Unclosed loop in program
8	Invalid context

Example Program

Command	Mode	Purpose
`P90 prog`	Header	Initialize program ID=1
`G91`	Header	Set relative positioning
`P29`	Header	Enable infinite looping
`M03 SY6 Y+`	Body	Start Yaw spindle (6 RPM)
`G0 SP30 P100`	Body	Pitch movement
`G0 SP30 P-100`	Body	Return pitch
`P92`	Body	Finalize program