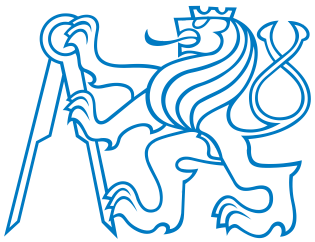
Czech Technical University in Prague

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 TODO 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

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: TODO

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: TODO

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
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 surveillance radar based on FMCW (Frequency Modulated Continuous Wave) technology. Goal of this thesis are to analyze SiRad Easy© radar system, determine its capabilities and construct a surveillance radar system based on this technology. This involves both hardware, in a form of two axis rotary platform, and software development be it for driving the rotary platform or processing radar data.

FMCW is a radar technology that uses a continuous wave signal whose frequency is modulated over time. By mixing sent signal over received a harmonic components are created whose frequencies are proportional to the distance of the target [1]. Thus by applying a Fourier transform to the received signal, individual distances can be determined. As opposed to pulsed radar, FMCW is able provide accurate distance measurement with relatively low power consumption [2]. 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[3]. This allows a relatively larger range of applications with ranges from couple of meters to up to 400 m [4] under ideal condition. However as SiRad Easy© offers rather low sampling rate and uses relatively low speed micrcontroller on board it is not suitable for any kind of 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 [5, 6] 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 radar and platform then are processed and visualized using MATLAB. While processing pipeline is relatively standard and uses common techniques such as FFT (Fast Fourier Transform) and CFAR (Constant False Alarm Rate) [7]. However a number of steps can be turned on and off, or their behavior modified by the user allowing a gread deal of flexibility. The data are then stored in radar cubes, common structure used in such applications [7]. Processed 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

Unlike classical Continuous Wave (CW) radars, Frequency Modulated Continuous Wave (FMCW) radars do not broadcast a signal at a single frequency. Instead, they employ a linear frequency sweep across a defined range. This approach enables range estimation without requiring pulsed transmissions while still allowing speed measurements using the Doppler shift. However, velocity calculations in FMCW radars are more complex compared to single-frequency CW radars.

The "MW" suffix in FMCW radar denotes that the system operates in the microwave frequency range. These high frequencies allow for compact antenna arrays, even enabling on-chip integration. Additionally, the millimeter-wave (MMW) portion of the spectrum is typically license-free [8] and offers large bandwidths, reducing the 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 electromagnetic signals, measuring the time taken for the signal to reflect back. In such systems, speed can be determined using the Doppler effect as

$$v = \frac{f_{\text{dop}} c_o}{2 f_{\text{rad}}}, \tag{1.1}$$

where f_{dop} is Doppler frequency, c_o is speed of light and f_{rad} is frequency of the radar signal. Distance is derived from the time of flight t of the signal as

$$d = \frac{c_o \cdot t}{2}. \tag{1.2}$$

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 must remain high, regardless of the number of pulses [2].

Maintaining high average transmission power poses legal and technical challenges. It increases the risk of interference with other devices and demands bulky, high-power circuitry – often requiring 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 the frequency sweep and the Doppler shift contribute to frequency changes in the received signal.

1.2 Basic principles of ideal FMCW radar

Let us picture an ideal FMCW radar system sending a periodic chirp with frequency frequency sweep from f_{c} to $f_{\text{c}} + BW$, so called sawtooth waveform. Other

FMCW systems may use a different modulations such a 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. Especially in case of triangle when multiple targets are present [2]. SiRad Easy© kit technically uses a segmented linear frequency modulation due to its limited computational power [9] but for the sake of simplicity we will stick to the ideal sawtooth waveform.

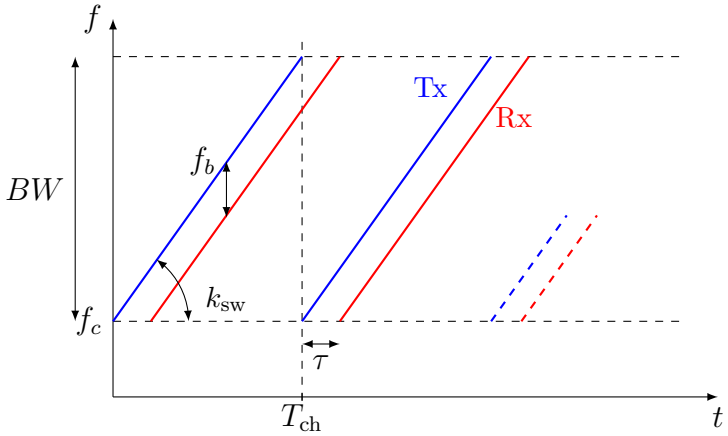


Figure 1.1: Ideal relation of frequency and time for received and sent signal

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

Let us define the chirp slope k_{sw} we can describe the change in frequency of the received 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 goes from 0 to chirp length T_{ch}. Standard equation of FM 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 amplitude of the signal, ω_c is carrier frequency and $f(s)$ is frequency of the signal. Substituting (1.3) into (1.4) we get the signal borrowed from the radar

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

Signal bounced back from the target will have the same equation with the only difference being 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

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

First additive term will lead to a signal with very high frequency, well above $2\omega_c$, this term doesn't carry any useful information and is usually filtered out – either by low pass filter or the frequency mixer itself [1]. Second term is so call beat signal whose frequency is directly proportional to the time delay τ. Applying a first time derivative to the cosine argument we get the frequency of the beat signal.

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

Calculating the distance to the target is now trivial, delay τ is equal to the time it takes for the signal to travel to the target and back

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

By substituting (1.8) into (1.9) we get the equation for distance

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

1.2.1 Limits of Range Measurement

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

However in most case the limit will imposed not by T_{ch} respectively BW but by 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 maximal distance of

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

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 spectrum is only inversely proportional to the chirp length and doesn't have any relation to sampling frequency [2]. Now we can enter 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.

There are of course other effect impeding the resolution of the radar system – such a phase noise around targets or sweep nonlinearity. Sweep Linearity can be both in the ramp itself – leading to decreasing resolution with range (with both linear and quadratic errors present) [1] (Appendix D and E). Or in sweep recovery (time to return to the start of the sweep) which leads to a fix decrease in resolution [10]. Both are however largely compensated in modern radar systems by using a closed feedback loop [1].

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 the target and v is the radial speed of the target. Within a single chirp there is no way to distinguish between the effects distance and speed of the target – thus multiple chirps are needed. Rewriting 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 withing a single chirp ($0 \leqslant t_{\text{s}} \leqslant T_{\text{ch}}$). Substituting (1.15) into low frequency part (1.7) leads to very complex equation, however according to [11] most of the terms can be neglected leading us to

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

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

In order to calculate Doppler shift frequency

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

where f_c is the center frequency, 2D Fourier transform can be used. At first FFT is run on each individual chirps and then another FFT is applied to these individual outputs in order ot determine phase shift. Given that FFT spectrum is calculated from complex data both halves of the specturm contain useful information – after applying `fftshift` we get 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 second distance of the target [11]

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

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

Maximal speed is dictated by how large of a phase shift can we measure without ambiguity – that is $\pm\pi$. From (1.16) we can see formulate the upper limit as

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

according to [12].

1.2.3 Radar equation

Fundamental equation to all radar systems is so called radar equation. This formulate takes into account relation of transmitted/received power, RCS of the target and other system parameters to calculate the maximal range of the radar. Written as

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

where P_{CW} is average system power in watts, G_{T} and G_{R} gain of transmitting and receiving antennas, σ_T is radar cross section of the target, λ is wavelength, L is system loss, k is Boltzmann's constant, T is effective temperature of the system in Kelvin, F_{R} is receiver noise figure, B_{Ro} is receiver bandwidth and SNR_{Ro} is required signal to noise ratio [2].

Knowing all parameters, or calibrating the system based on know reference, we can calculate target's RCS. This value gives us much clear understating of the target as opposed to just displaying power received or its logarithm as the RCS is a constant value for a given target, regardless of the distance [7].

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 module both based on ICs from Indie Semiconductor (TRX-024-007 and TRA-120-001, respectively). Both are strictly SISO IC with two antennas – one for receiving another for transmitting so azimuth estimation is not possible by default.

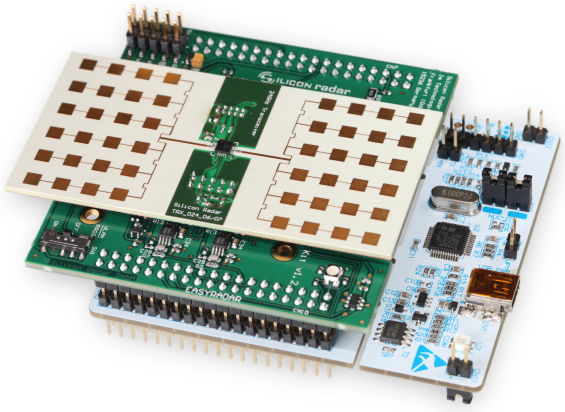


Figure 2.1: SiRad Easy© 24 GHz configuration

Direct communication with the radar board itself 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 over on board ESP32. Both are relatively low bandwidth communication – the serial maxing at baudrate of 1 000 0000.

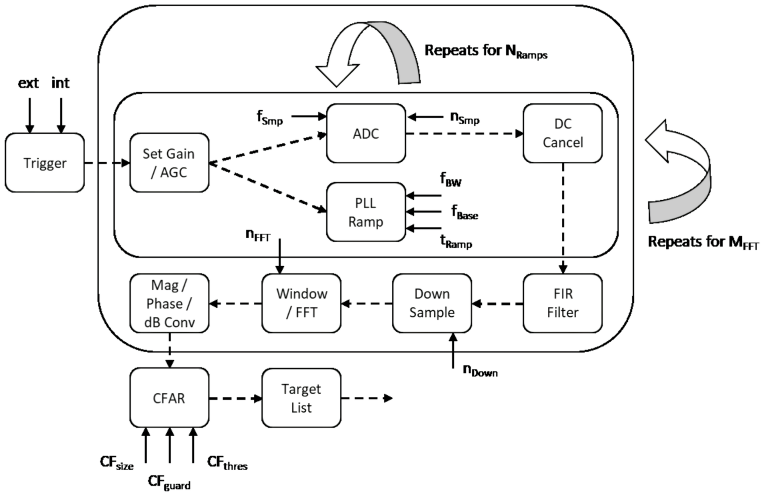


Figure 2.2: Flow of Radar Measurement on SiRad Easy©

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 [9]. What happens to the sampled ramps remains unclear, as the manufacturer has not provided details. It is likely that the data are somehow averaged to improve the SNR of the reported signal. As only N_{samples} are reported by the radar but according to manufacturer provided equation (2.1) each ramp should be sampled with that number of samples. Additionally, due to the devkit's limited computational resources, the radar does not emit a continuous sawtooth waveform but rather a segmented one [9].

Unfortunately whole devkit is clearly designed primarily for range detection with little to no regard payed for speed measurement. Aside from need to turn of averaging of the ramps radar doesn't maintain constant intervals between chirps. This is the case even when relying on manual triggering of the radar, relying on self trigger yields even worse results. Not to mention radar doesn't provide any information about timing.

When measuring reporting interval with 20 ms trigger radar achieved average time of 20.5 ms with standard deviation of 1.3 ms. However from time to time they were outlier when report came in only after 30 ms in worst case or 16 ms in best case measured on 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 same triggering interval of 20 ms, the average time between chirps was 20.7 ms with standard deviation of 0.7876 ms.

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.

Per radars manual [9] 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 even with low 8 NFFT steps the resolution is only 0.031 ms^{-1}.

2.1 Outline of 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 detailed analysis of the radar system and enables the implementation of custom signal processing algorithms. For these reasons, the default windowing and undocumented filtering was also disabled. As for trigger as previously stated self trigger isn't that consistent and thus manual trigger sent over UART was used.

After configuration of the output regimes, it is necessary to find the correct balance between the number of ramps, samples, and sampling frequency. Since the radar will be in motion, frequent updates are crucial to ensure accurate reporting. Faster updates allow for higher platform speeds or make it possible to use neighboring samples to aid in evaluation of current one (As the angular distance between them remains relatively small.).

Even with irregular chirp intervals rough speed estimation can still be done. As deviation isn't that large the error of using standard FFT should still be within acceptable limits. If not there is the possibility of easily swapping to non uniform DFT and basing chirp intervals in reporting intervals. For any speed estimation to be possible however the number of ramps withing a group must be fixed at one. As any averaging would destroy information about Doppler shift.

Another parameter to consider in relation to rotary application is AGC. Using it adds two additional ramps that are used solely to set gain value but it also can lead to inconsistent weights of values between two neighboring data sets. Former only introduces a small slowdown but the letter could be a problematic, especially since radar doesn't really report change in AGC value. Thus its safest to turn AGC off and set the gain manually depending on how the readings appear in post 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 a considerable amount of time, and the radar does not provide any feedback to indicate when a new configuration can be safely applied. As a result, commonly used techniques such as alternating chirp slope or frequency modulation are not feasible.

Regarding 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 is the ease and speed of parsing. Even though MATLAB is not particularly optimized for parsing binary data, it still processes binary output approximately 40% faster than human-readable TSV data.

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), with an additional ultra-wideband mode supporting 23–26 GHz [13]. On the SiRad Easy© there is

no distinction made between those two modes and the user is free to set any bandwidth [9] . The transmitter output power ranges from 2.5 dBm to 6 dBm, depending on the configuration [13]. A maximum range of 400 m is advertised [4], though this is likely under ideal conditions when observing a large target. Also as previously stated the radar will operate on much lower sampling frequency than needed to achieve such results.

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 [13].

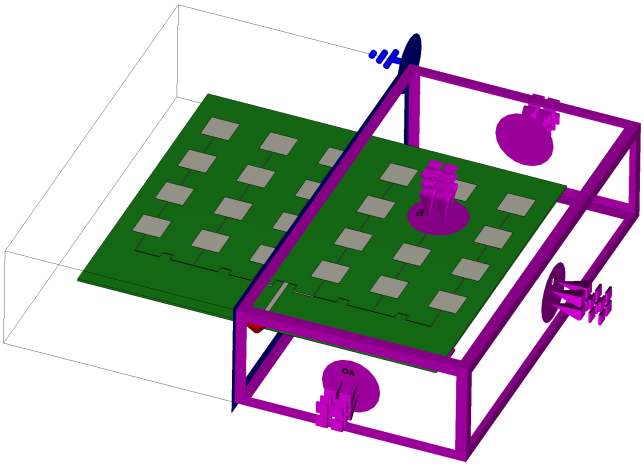


Figure 2.3: Simulated 24 GHz header with boundary conditions showed

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 23A.478 GHz with second minimum at 24.518 GHz (Figure 2.4). Lack of minimum at 24 GHz may be attributed to measurement difficulties (The etching quality of the copper traces was suboptimal.) and the neglecting variations of substrate parameter due to interactions of different 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), 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.

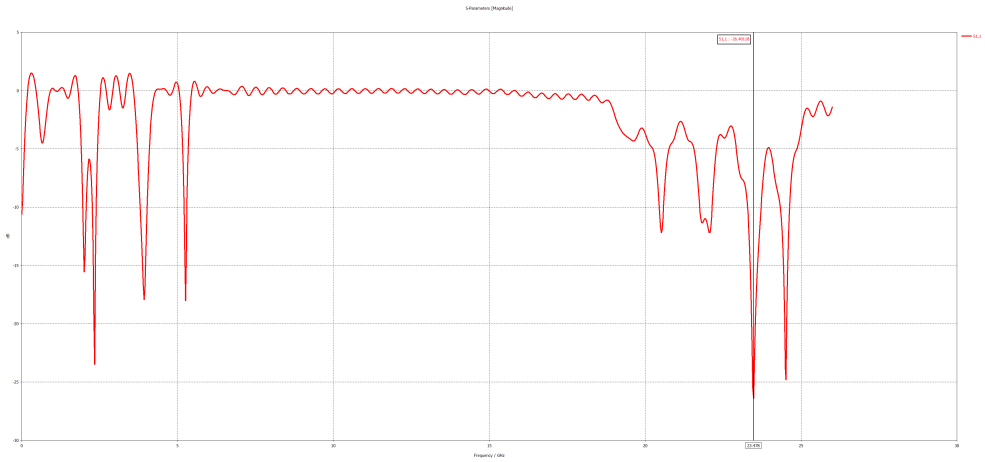


Figure 2.4: s_{11} parameter of the 24 GHz header

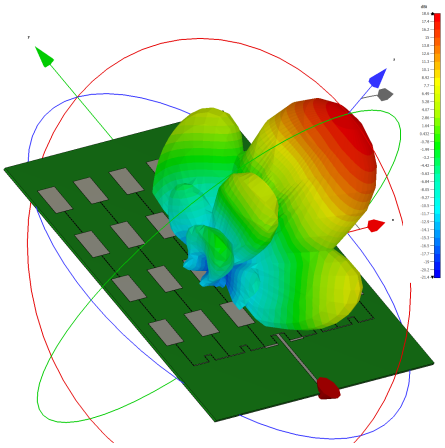


Figure 2.5: Radiation pattern of 24 GHz header – 3D view

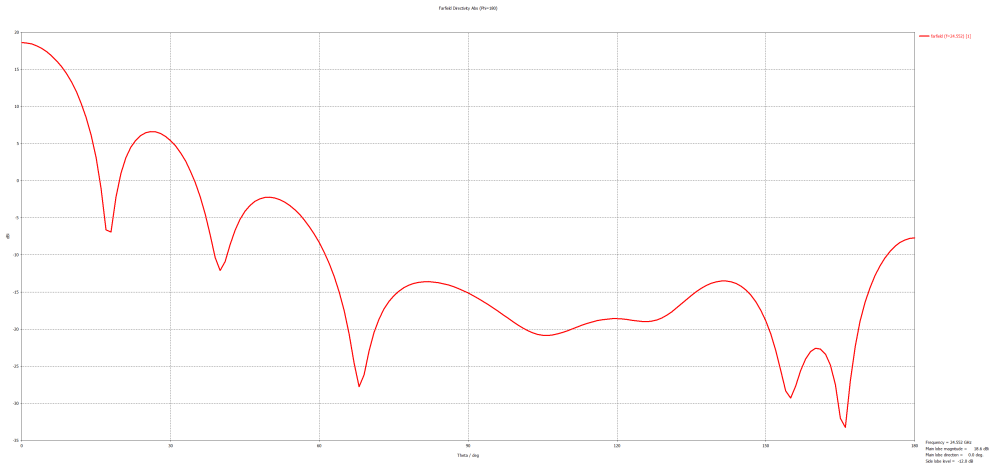


Figure 2.6: Radiation pattern of 24 GHz header – 180° norm

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

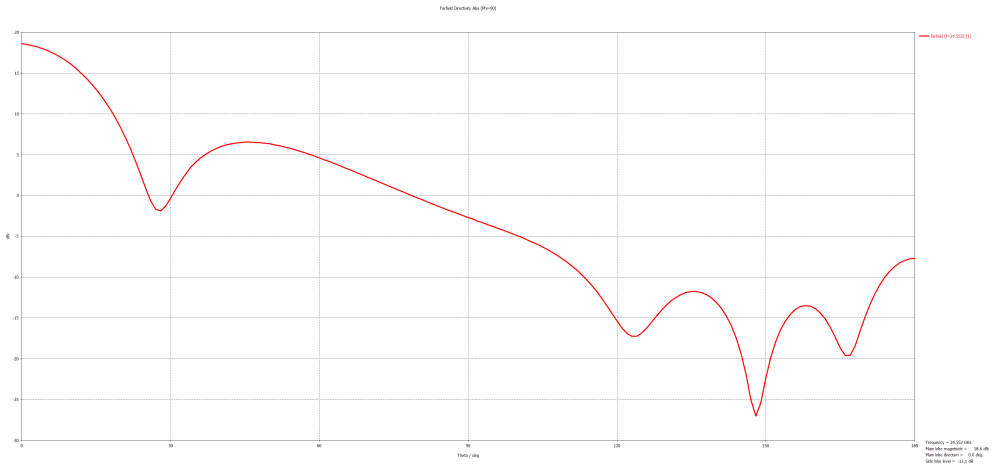


Figure 2.7: Radiation pattern of 24 GHz header – 90° norm

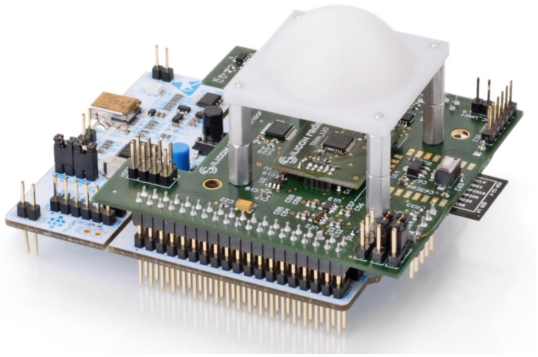


Figure 2.8: SiRad Easy© with 122 GHz header

band, with output power ranging from -7 dBm to 1 dBm [14]. The chip is capable of detecting large targets at distances of up to 40 m [4].

Out of the box performance of the system is quite bad (Figure 2.9) with width of the main lobe being roughly ±40° in both E-plane and H-plane [14]. However, this can be significantly improved using the supplied collimator lens, reducing the main lobe width to ±4° [15] (see Figure 2.9).

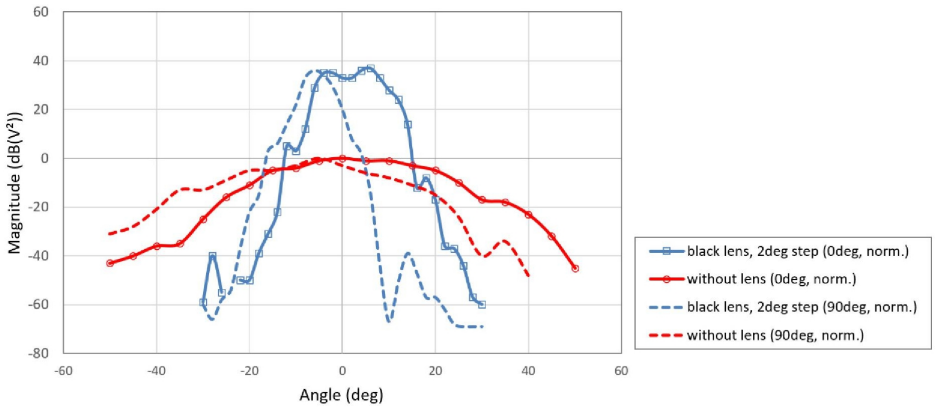


Figure 2.9: Radiation pattern of 122 GHz header comparison

3. Rotary Platform

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

3.1 Platform Design Parameters

To begin, it is essential to outline the fundamental requirements for the platform. These stem from the physical capabilities of the SiRad Easy© and need to provide easy to use interface to control the platform.

3.1.1 Physical Capabilities

The primary constraints on the physical design arise from the radar's radiation pattern (considering both the 24 GHz and 122 GHz headers). These patterns determine the necessary clearance in front of the radar and, more importantly, the precision required for platform movement. As stated previously, the 24GHz radar has a main lobe width of approximately $\pm 7°$, while the 122 GHz radar has a main lobe width of $\pm 4°$ [14]. To minimize strong reflections from both the main lobe and side lobes, a conservative clearance of $\pm 45°$ in front of the radar was selected.

Due to the relatively low angular resolution of the radar, high platform precision is not required. A basic 200-step stepper motor with a step size of 1.8° is sufficient; however, smoother motion simplifies software compensation of the movement. Given the radar's low weight—measured at 120 g, including the mounting bracket, smoothness of movement can be achieved solely through motor microstepping.

High-speed movement is unnecessary for this application. The manufacturer specifies a maximum update frequency of 50 Hz, corresponding to a new measurement every 20ms [3]. In this use case, the update frequency will be closer to 10-20 Hz. Using

$$t_{\text{angle}} = \frac{60}{360 \cdot N_{\text{RMP}}} \cdot \alpha, \tag{3.1}$$

where t_{angle} represents the time spent traveling an angle α in seconds, and N_{RPM} is the number of rotations per minute, we can calculate that even at a low rotational speed of 60 RPM, an 8-degree movement (matching the angular width of the main lobe for the 122 GHz radar) takes only 10 ms – too fast to properly interpret the data.

3.1.2 Software Requirements

Given its widespread adoption as an industry standard for controlling multi-axis 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, changing units from steps to degrees or enabling automatic homing. Specifically for surveillance radar application spinde

regime, where the radar is constantly rotating without any new commands, is needed. This should be done with a degree of control that what is typically offered by G-code interpreters where spindle regime requires special configuration and on the fly can usually only be turned off and on [16]. 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.

3.2 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. This 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 [17]. More problematic than loss is cross talk between signal wires when stepper motor responsible for tilting the radar is running. USB 2.0 connection doesn't seem that effected, however endstop used for homing requires adding a low pass filter to its output to help with noise.

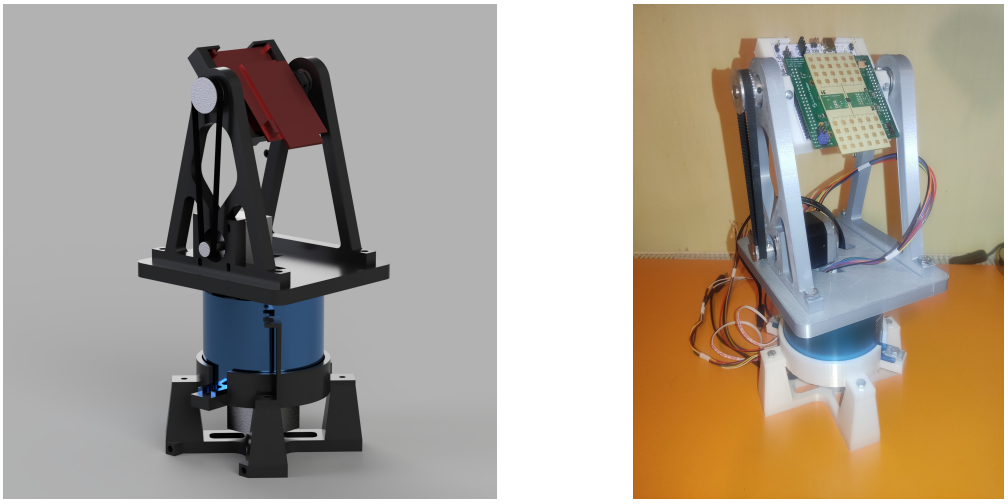
Unfortunately, the manufacturer opted for a non-standard male-male USB 2.0 connection, requiring a female-female adapter to connect the radar to the slip ring. Additionally, when used with a poor-quality cable, the system exhibits signal integrity issues. These problems could likely be mitigated by integrating a signal conditioner into the transmission line.

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

The rotating section features a simple A-frame design that elevates the radar, which is mounted on bearings, allowing it to tilt freely. To control the tilt, 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 mounted on the second support strut.

Since 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 33 cm in height and has a footprint of 20×20 cm.

While the platform offers some ability to tension the belt in currently printed configuration the range is not sufficient to fully tighten the belt. While slippage is not a problem radar there is some slack in radar's tilt. However as the radars capabilities in this direction aren't particularly high it was decided not to reprint the whole assembly. Still in supplied 3D models these issues were fixed.



(a) 3D render (b) Photo

Figure 3.1: Form of the final assembly

3.2.1 Platform Electronics

The electronic aspect of the platform is relatively simple, involving only two main tasks: 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 is sufficient. For this purpose the A4988 stepper driver was chosen due to its low cost, microstepping capabilities and basic current control [18]. A minor drawback is the lack of feedback from the driver to the microcontroller, including the absence of 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 complementary kits. This might be necessary in application where platform loudness would be a serious problem as A4988 produces rather noisy driving signal leading to even 20 dB higher operating volume [19].

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 [20]. 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. However, since the system does not require specialized peripherals or high processing power, any ESP32 variant would be sufficient.

3.3 Platform Software Realization

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 an 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 the next is primarily limited by the inertia of stepper motors and not by the software.

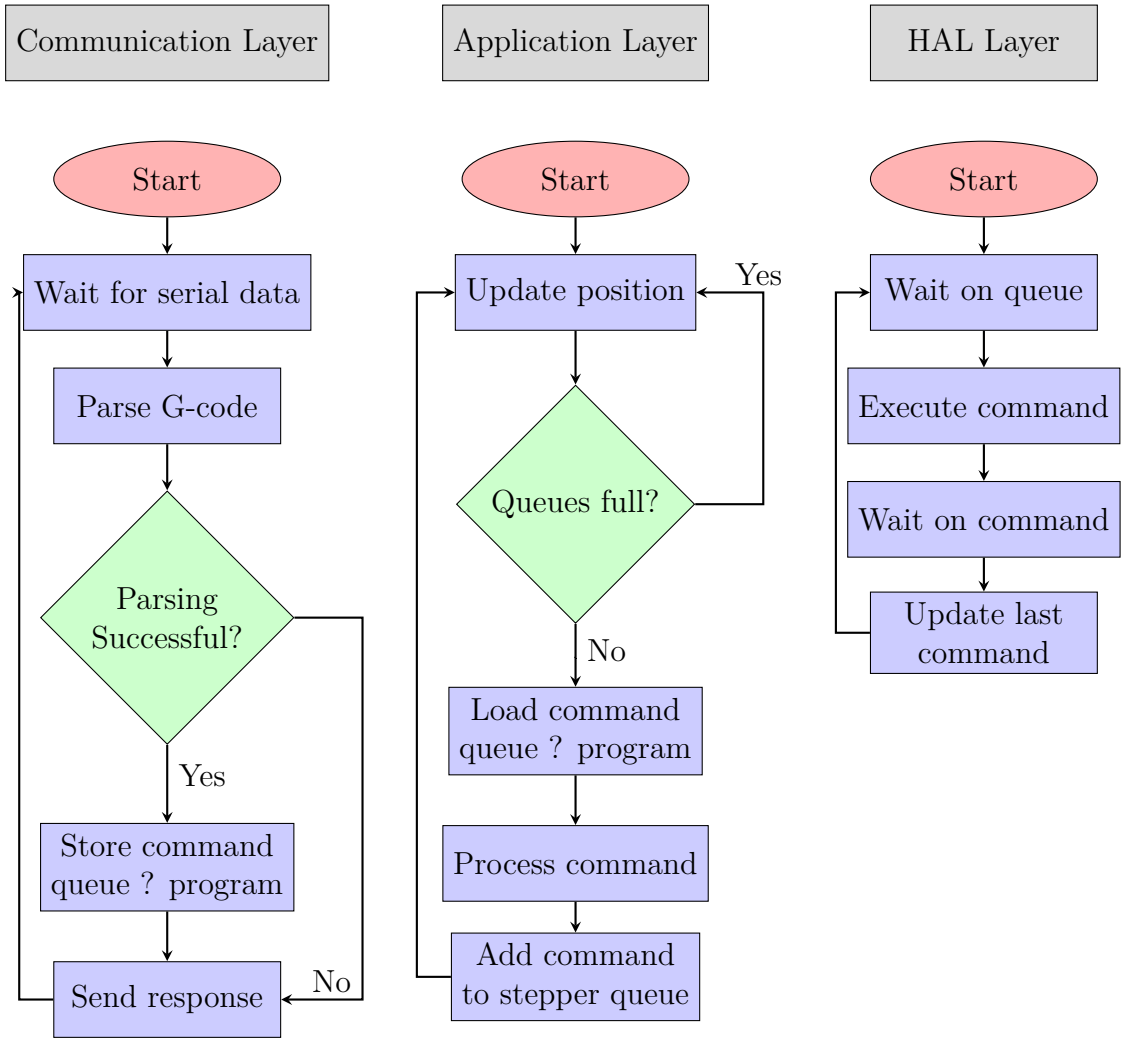


Figure 3.2: Program flow diagram

Also 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 validate current setting and is expected to be only operated from computer application, which can handle this and provide user

with feedback.

3.3.1 Communication layer

The communication layer manages incoming data over the serial line, with efficient handling facilitated with the aid of RTOS queues. Upon receiving data the text string is parsed and either pushed to a queue or added to programm declaration, in case we are currently declaring program.

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

3.3.2 Application layer

The application layer performs two primary functions: tracking the 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 make necessary calculations to facilitate absolute positioning and enforce movement limits.

A key departure from standard G-code interpreters, like [16], is how the platform handles single-axis move commands. When a move command targets only one axis, the other axis remains free to read next command and begin its execution. If this behavior is undesirable, the user must issue commands for both axes. 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 is the requirement for separate positioning modes for each axis. Continuous rotation prevents calculations of 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 settings.

In order to support or possible operating regimes 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 other application that require real time control of the platform. However in this regime no limits are enforced and the platform operates strictly in relative positioning mode.

3.3.3 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 calculations are handled in the application layer whole routine remains highly efficient.

The main challenge lies in generating precise PWM signals (Used to control stepper motors drivers.) and stopping signal generation after a specific number of steps. Using the equation:

$$t_{\text{delay}}(s) = \frac{60}{2 \cdot N_{\text{steps}} \cdot s}, \tag{3.2}$$

where s is speed in RPM, N_{steps} is the number of steps (Anywhere from 200 to 1600 depending on microstepping.), and t_{delay} is the time between steps, we calculate that even at 30 RPM, the delay between output changes is 5 ms per step. With microstepping at a 2:1 ratio, this reduces to 2.5 ms – faster than lowest sleep interval on ESP32 and without sleeping the RTOS watchdog will trigger. Therefore, signal generation must leverage specialized microcontroller peripherals.

The ESP32 platform offers two options: 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 [21], synchronization is restricted to its proprietary API, and there is no straightforward way to track progress during a move [22].

For these reasons, MCPWM and PCNT were chosen. MCPWM handles pulse generation, while PCNT counts steps, enabling easy synchronization, continuous rotation, and a robust API for step tracking [23]. The only limitation is the PCNT's 15-bit counter, which caps the maximum steps per move at 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.

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

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

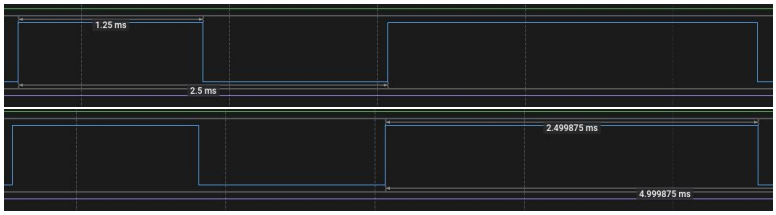


Figure 3.3: Moment of change between commands (120RPM ⇒ 60RPM)

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. Most importantly, the platform's timely and predictable behavior ensures that mathematical corrections to the radar data can be applied accurately.

4. Control application

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

4.1 Main application interface

Upon starting the application user is presented with the 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 in case of an emergency stop the platform. Lastly topbar is used to access preferences and platform control interface.

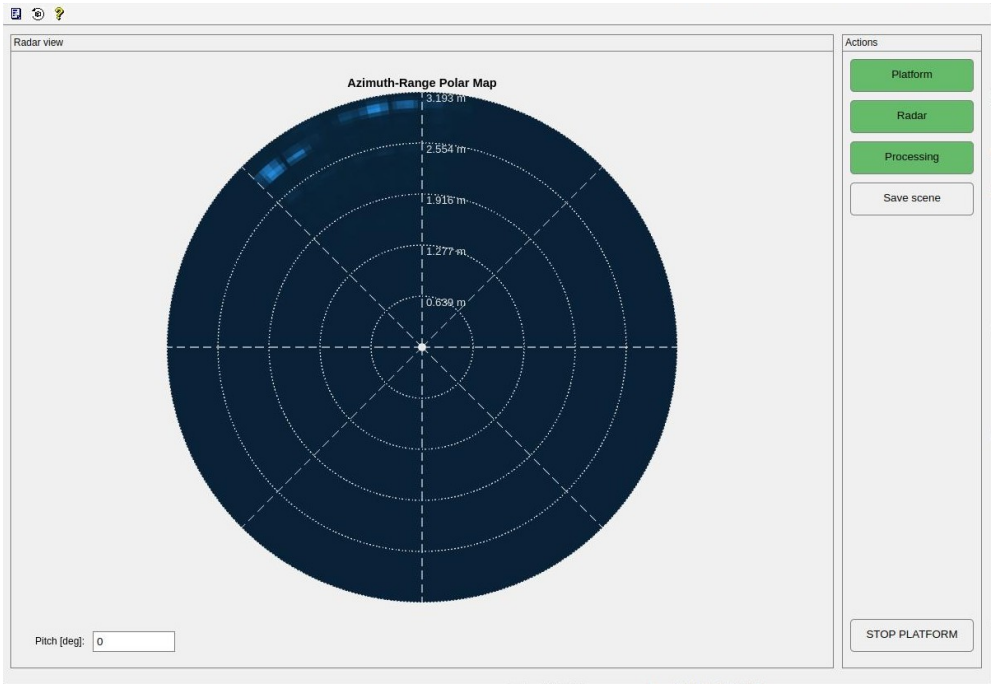


Figure 4.1: Main application window

4.1.1 Configuration options

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 in given application. Firstly – most limits imposes by the GUI on user chosen values are guided more by mathematical requirements and what configuration allows (In case of SiRad related options.) than by what would be practical.

Secondly – configuration of components such a CFAR or DBSCAN strongly depended on bandwidth used, respectively width of a single range bin, among other things. CFAR guard parameter of 3 might be too small on 5 GHz bandwidth where width of single range bin is only 50 mm, but if used on 24 GHz header with $BW = 500$ MHz where range bin width is 50 cm 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. However baudrates are usually fixed – 230400 or 1000000 for the radar, depending on the firmware used and 115200 for 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.

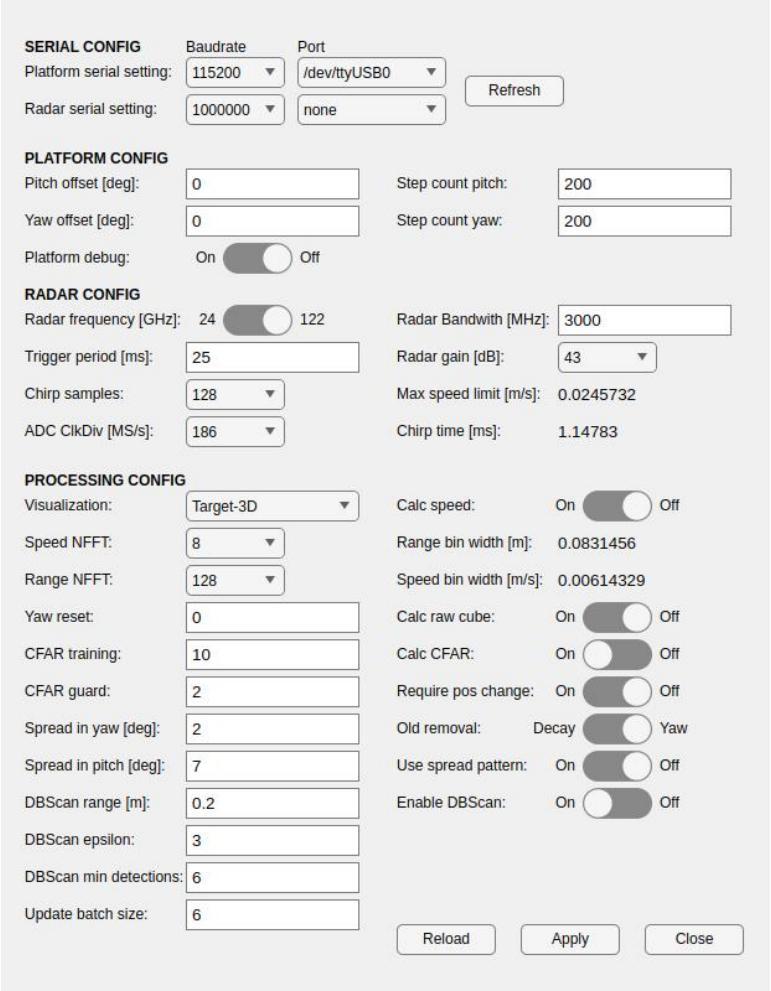


Figure 4.2: Preferences menu

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 and editable 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 and 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 platform program comprised.

In preferences menu 4.2 user can configure static offsets to the platform with the `Pitch offset` and `Yaw offset` options. This is useful in case zero positionof 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 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.

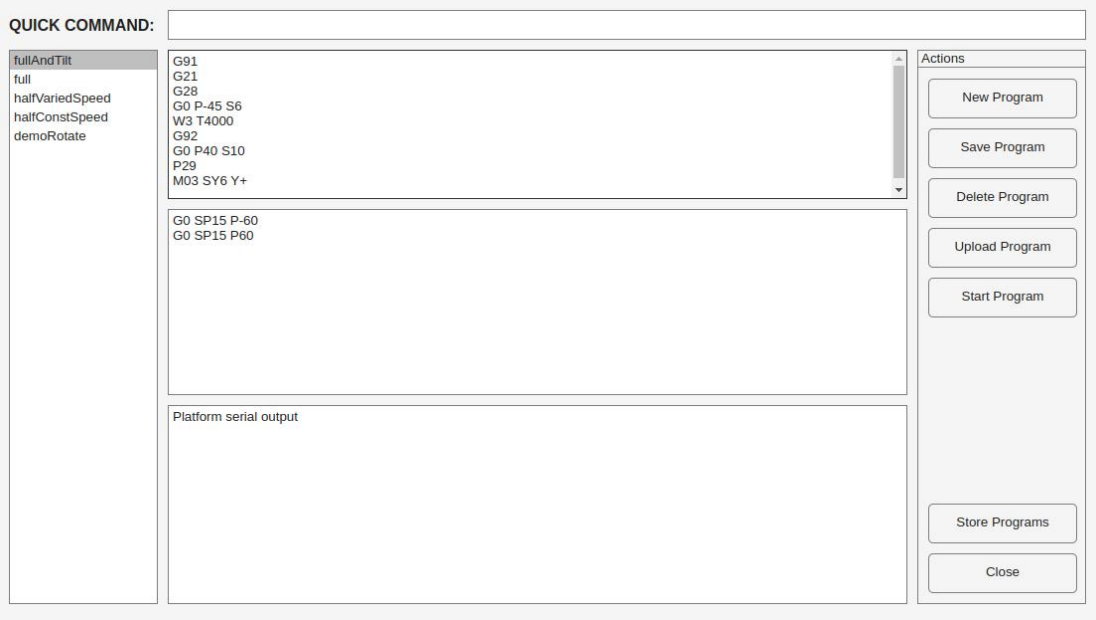


Figure 4.3: GUI of the platform control

Managing platform programs is done via a separate window (Figure 4.3). This interface allows 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 in G-code like syntax. 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 `Yaw reset` with tolerance zone fixed at two degress on either side. In orde to prevent multiple emission of the event the platform will only emit it once per 1 s. This event allows rest of the code to delete stored data and start gathering new one in next cycle.

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 (figure 4.2) 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 [9], in practice, especially with the 24 GHz header, achievable bandwidth was usually much lower.

User can also configure the 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 correct signal to noise ratio.

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 with slightly more stable performace at some 22 ms.

5. Radar Data Processing

Firstly it is necessary to outline what inputs and outputs of the whole processing pipeline are. Data to the applications 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 a data is received. Processing than does user chosen calculations – parts such as CFAR calculation, speed calculation and so on can be turned on or off. Output data are then stored in radar cube, common method to manage radar's data that enables easy integration of more complex algorithms in the future [7].

Given that both CFAR and raw data (That is individual range-Doppler maps.) calculation might need to be stored application has two 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 rangeing 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 guided by attempt to perevent radar from seeing behind itself.

Main complicating factor in the processing was the need to keep main thread responsive. Most time critical thing main thread does is timely readout of data at serial ports. If data can't be read right after its arrival assigned timestamps will be incorrect and visualization will be wrong. In addition GUI starts to be quite unresponsive if the resources of 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 and so is the case for certain file/memory mapping functions [24]. Thus the parallelization needs 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 data are buffered into a circular buffer.

2. Data processing: Data 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 in user chosen format.

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 data stream. After whole frame from the radar is read, the 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 for an complex singla which is multiplied by Hamming window. Basic 1D FFT is then calculated with $NFFT$ equal to `Range NFFT`, 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 multiple FFT spectrums computed – those can be leveraged to enable range-Doppler map computation.

Program then 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 platform 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 check can be skipped by toggeling `Require pos change`.

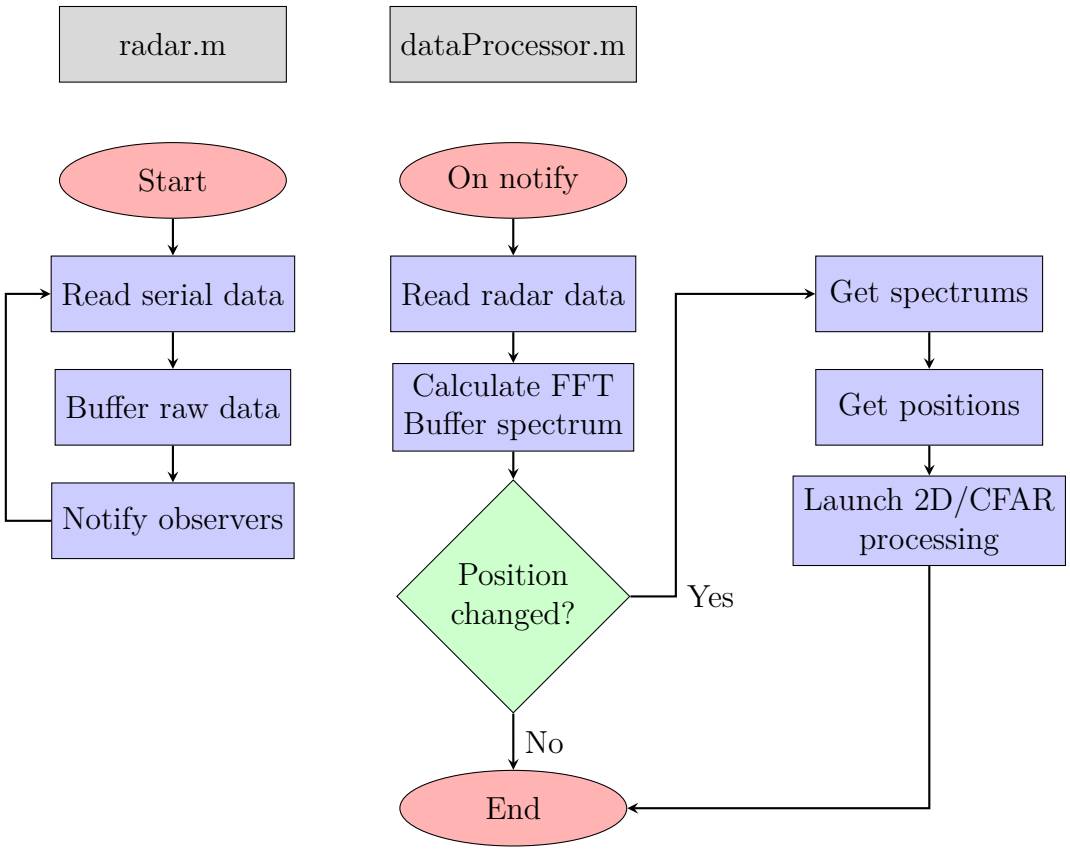


Figure 5.1: Processing flow - Data acquisition

5.2 Data processing

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

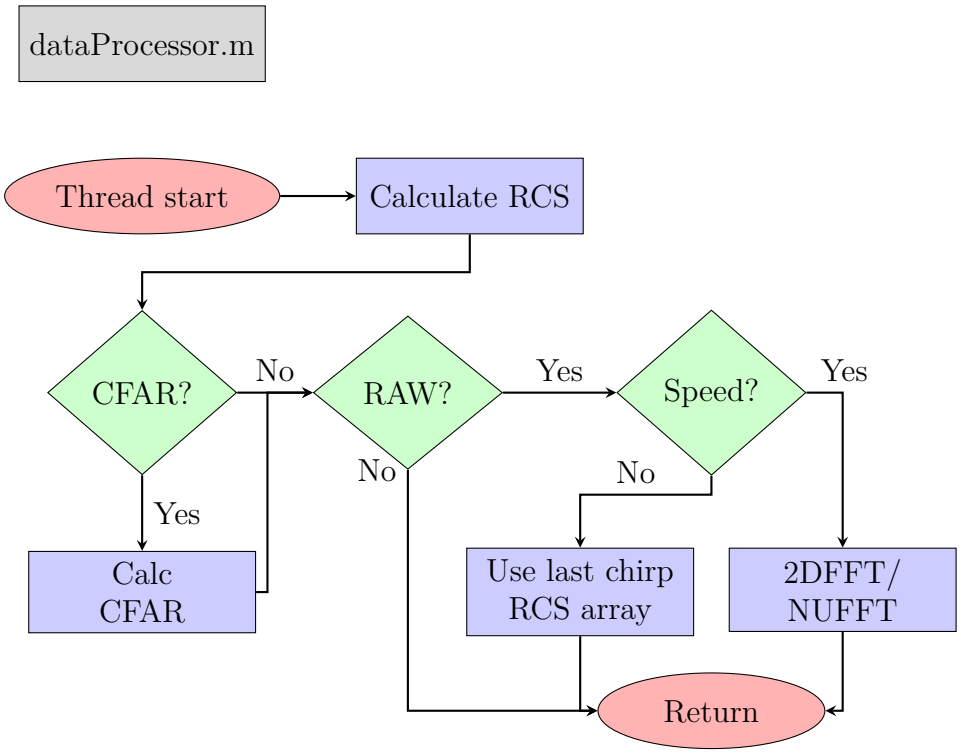


Figure 5.2: Processing flow - Data processing

In the first step last spectrum is converted to values proportional to the radar cross section (RCS) 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 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 will be runned on latest chirp in the batch. CFAR is a simple algorithm that enables basic target detection in the radar data. This is done using adaptive thresholding when value at each bit is compared againts mean of values in its neighborhood which is separated from bin under test by a number of guard cells [7]. Algorithm than simply detects if compared to its surrounding there is a peak in the data suggesting a targer is there. In this application simple 1D 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.) [25]. Both these paramters are configurable from preferences menu with `CFAR training` and `CFAR guard`. As already stated, due wide varriation in bin

widths for different bandwidth both paramters are left to be configured to the user.

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 on program firstly cuts of samples that are too far away from position of the latest chirp. After that timing of chirps is also analyzed choosing wheter NUFFT or 2DFFT transformation should 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 ration is over 20 % NUFFT is used. For more accurate measurement more complex method should have been used but as the radar timing data is 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 rotation of the platform is done and the threshold for distance isn't user configurable. 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. 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 filling the buffer 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 [26]. 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 cubs contating either CFAR or raw data, are quite simmilar, in case 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 whole cube is multiplied by decay factor that is derived from platforms speed of movement (The slower the platform moves the slower the decay). Using this option will lead to output more visually resembling radar output of 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 values 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 around 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.

Enabling spread pattern leads to degradation of quality of the data in exchange

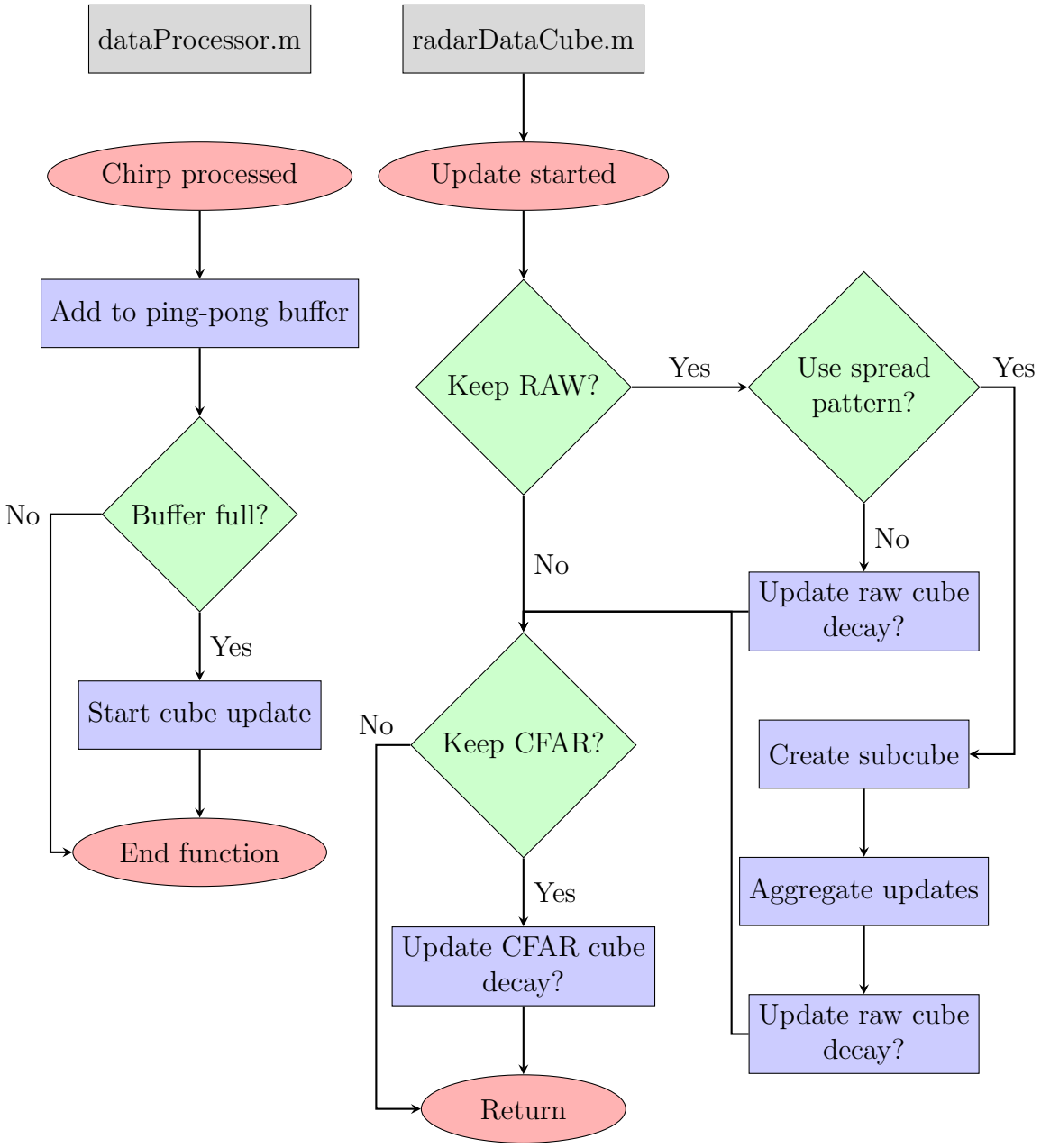


Figure 5.3: Processing flow - Cube update

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 a aid of 2D matrix, containg a 2D Gaussian function. As the data in series are usually close to each other in terms of position instead of directly adding these contributions to the full cube much smaller cube is created that covers only the area needed for the batch. This enables significant speedup as smaller memory size enables better cacheing. After each update is aggregated to the subcube the original cube is update in one go.

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 perforace increase thanks to SIMD instructions.

5.4 Visualization

After cube update finishes program returns to the main thread and the data are visualized. Other than graphical access to the data is not possible. However the application offers a number of different visualization styles to suit the user's needs.

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. Which pitch angle is used is chosen via the small textbox at the bottom left corner of the screen. 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 5 GHz giving only some 3 meters of range.

As for the form of the data instead of calculation projection into horizontal plane the data are displayed in a conical projection. That is no additional calculations are are 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.

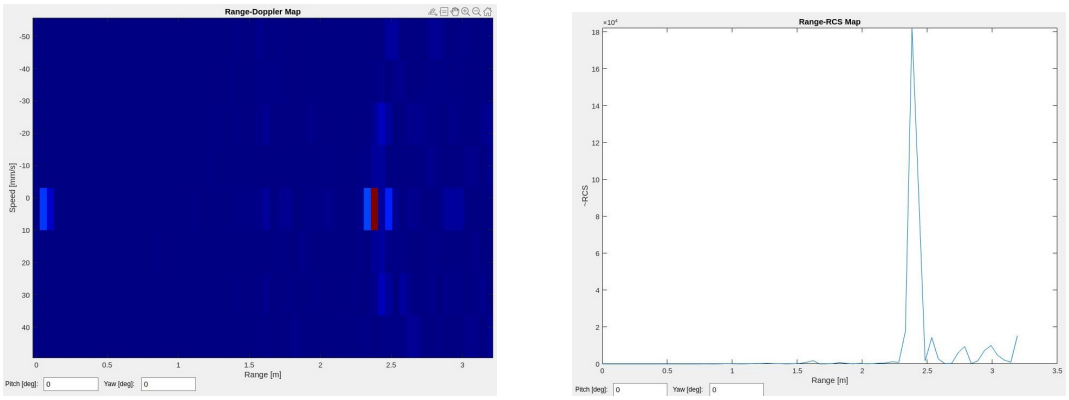
It necessary to point out that as 24 Ghz header radiation pattern in pitch is rather large at relatively low angles the radar provides information about a larger area of the scene. And if the user wishes using the spread patterns allows to range-azimuth map to include data even from different pitch angles.

5.4.2 Range-Doppler

Picking this visualization style from the preferences menu will display the data in a classical xy 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 taken.



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

Figure 5.4: Range-Doppler and Range-RCS maps

5.4.3 Target-3D

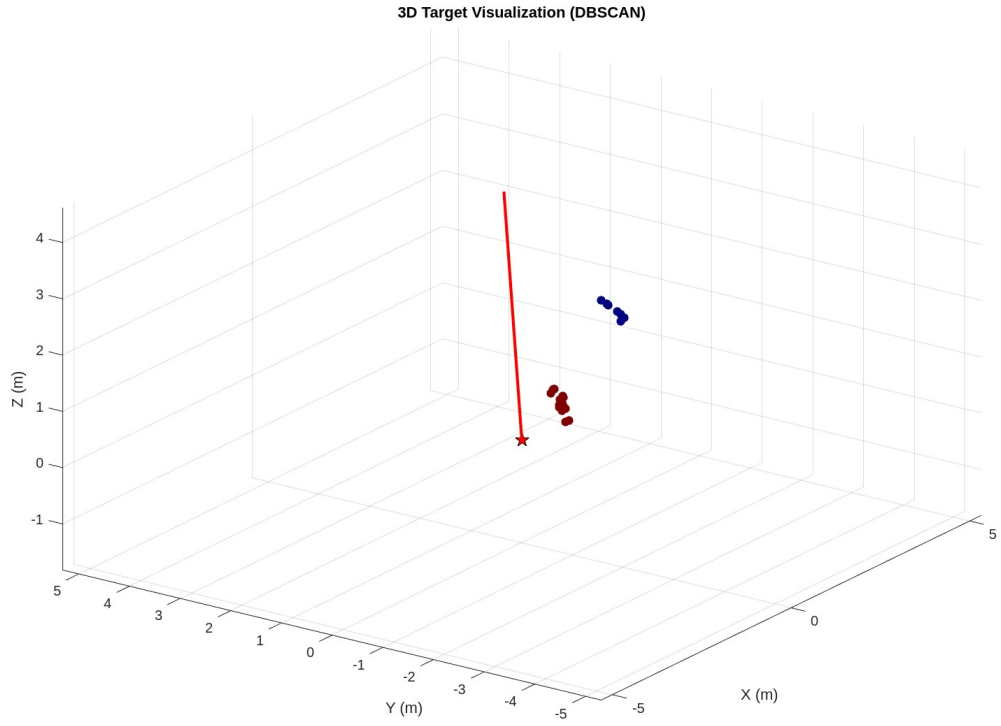


Figure 5.5: 3D Space visualization method

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 yaw and pitch of the platform at time of the update is also 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 [27]. Gerneral idea of DBSCAN is that to 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 a cluster centers [28]. In MATLAB's in algorithm distance threshold and minimal number of points are both configurable variables with other input being just the points list [27]. 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. 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. Both these factors lead to problems with choosing both N and ε paramters.

For accurate results these problems can be overcome using many different methods like using adaptive thresholds in relation to radius [28] or relying on more clever axis systems then just polar or cartesian [29]. However as in this thesis as DBSCAN is more of an illustration of possible postprocessing that can be added to existing base no such complex methods were implemented. Both ε and N parameters are left to the user to configure. Distance between points is the firstly converted to cartesian coordinates then metric is calculated using standard euclidean formula after which is later divided by average radius of the two points. Still it needs to be said that, while dividing by the mean of distances is sometimes used when working with spherical data [30], author is not aware of any literature that would suggest use in DBSCAN. Especially on longer distance this methods will lead to significant errors.

5.5 Scalability of the processing pipeline

Whole processing is written in a way that if radar module was exchanged for a faster one capabilities could be much more extended while keeping most of the codebase similar. Aside from different function reading the data from the radar all code is relatively lightweight till the point position change is detected and multiple fft spectrums are being processed.

Thus radar with just increased chirp frequency would only lead to better speed recognition. Especially if platform movement remained the same. In case of increase of platform rotation possible corrections would need to be done in the processing pipeline.

In case number of FFT points both in speed or range would be significantly larger that what has been tested (Maximal cube dimensions 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. Specifically the decaying of the cube would be possible for much larger dimensions. In case of solely CPU based processing decay functions probably would be impractical for much larger cubes and 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.

Conclusion

TODO

Bibliography

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

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

[3] Indie semiconductors GmbH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: User Guide*. 2019. Version 2.1.

[4] Indie semiconductors GmbH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: User Guide*. 2019. Version 2.1.

[5] *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.

[6] *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.

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

[8] *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.

[9] 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.

[10] 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.

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

[12] 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/.

[13] 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.

[14] 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.

[15] 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.

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

[17] *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.

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

[19] 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/.

[20] 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.

[21] *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.

[22] *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.

[23] *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.

[24] *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.

[25] *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.

[26] *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.

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

[28] 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.

[29] 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.

[30] 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.

5.5.1 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

5.5.2 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