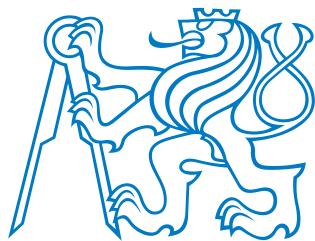


Czech Technical University in Prague

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

DECLARATION

Já, níže podepsaný

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

prohlašuji, že jsem bakalářskou prací s názvem

Pohledový FMCW radar

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

Prohlašuji, že jsem v průběhu přípravy a psaní zálohy na práce použil nástroje umělé inteligence. Vygenerovaný obsah jsem ověřil. Stvrdzuj, že jsem si v domě, že za obsah zálohy na 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 radar, ESP32, MATLAB, Číselné zpracování signálů, 3D tisk, G-code

Contents

Abbreviations	1
Introduction	2
1 FMCW Radar Fundamentals	4
1.1 Comparison FMCW Radar to Pulse Radar	4
1.2 Basic principles of ideal FMCW radar	4
1.2.1 Limits of Range Measurement	6
1.2.2 Speed Measurement	7
1.3 Radar equation	8
1.4 Angle of Arrival/Spatial Information	8
1.5 CFAR	9
2 SiRad Easy[®]	10
2.1 Outline of the Chosen Configuration	12
2.2 24 GHz Header	12
2.3 122 GHz Header	14
3 Rotary Platform	16
3.1 Platform Construction	16
3.1.1 Platform Electronics	17
3.2 Platform Software	18
3.2.1 Communication Layer	18
3.2.2 Application layer	19
3.2.3 HAL Layer	20
4 Control Application	22
4.1 Main Window	22
4.1.1 Application Configuration	22
4.2 Platform Control Interface	24
4.3 Radar configuration	25
5 Radar Data Processing	26
5.1 Data Acquisition	27
5.2 Data Processing	28
5.3 Cube Update	29
5.4 Visualization	31
5.4.1 Range-Azimuth	31
5.4.2 Range-Doppler	31
5.4.3 Target-3D	32
Conclusion	34

Abbreviations

Abbreviation	Meaning
AGC	Automatic Gain Control
CFAR	Constant False Alarm Rate
CW	Continuous Wave
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DFT	Discrete Fourier Transform
devkit	Development Kit
FFT	Fast Fourier Transform
FMCW	Frequency Modulated Continuous Wave
MCPWM	Motor Control Pulse Width Modulation
MIMO	Multiple Input Multiple Output
PCNT	Pulse Counter
RCS	Radar Cross Section
RMT	Remote Control Transceiver
SIMD	Single Instruction, Multiple Data
SISO	Single Input Single Output
SNR	Signal to Noise Ratio
TSV	Tab Separated Values

Introduction

The following thesis concerns the realization of a generic surveillance radar system based on FMCW (Frequency-Modulated Continuous Wave) technology. Conventional FMCW-based surveillance radars often employ multiple receiving antennas or are implemented as MIMO (Multiple-Input Multiple-Output) systems. In MIMO configurations, electronic beam steering is possible, enabling a larger field of view and the detection of targets in both elevation and azimuth. Simpler systems, which use only multiple receiving antennas, typically allow for azimuth estimation through phase difference analysis [1]. However, such systems generally require significant processing power and are not capable of providing comprehensive 3D spatial information due to their limited field of view.

To address these limitations, this work proposes a simpler approach: a radar system with just a single RX and TX antenna, where the beam is steered mechanically using a rotary platform. Initially, the capabilities of the SiRad Easy[®] evaluation board are assessed, followed by the development of a custom two-axis rotary platform. Both components are then integrated into a unified system, controlled and managed via a MATLAB desktop application. This integration involves both hardware—namely the rotary platform—and software, including control of the platform and processing of radar data. Similar systems have been previously developed – both those relying on a single axis of rotation [2, 3] or complex commercial solutions enabling surveillance of the whole 3D space [4].

FMCW radar operates by transmitting a continuous wave signal whose frequency is modulated over time. By mixing the transmitted and received signals, harmonic components are produced, with frequencies proportional to the distance of a target [5]. Applying a Fourier transform to the mixed signal enables the determination of object distances within the scene. Compared to pulsed radar, FMCW can provide accurate distance measurements with relatively low power consumption [6]. Velocity estimation is also possible by exploiting the Doppler effect. However it's nature is more complex than in pulsed systems.

This thesis focuses on the SiRad Easy[®] radar system—a low-cost evaluation board designed to familiarize users with FMCW technology, offering both 24 GHz and 122 GHz headers [7]. This versatility allows for a wide range of applications, with detection ranges from a few meters up to 400 meters under ideal conditions [8]. However, due to its relatively low sampling rate and the use of a modest microcontroller, the SiRad Easy[®] is not particularly well-suited for speed measurement: the maximum measurable velocities are well below one meter per second, and even then, measurement accuracy is limited.

To complement the radar, a custom rotary platform was designed and constructed for this thesis. While commercial solutions exist [9, 10], they are often prohibitively expensive and include unnecessary features. Therefore, a more affordable platform was built from off-the-shelf and 3D-printed components, controlled by an ESP32C6 microcontroller. This controller interprets G-code-like commands and drives the stepper motors.

Data from both the radar and the platform are processed and visualized within MATLAB. The processing pipeline follows standard approaches, utilizing techniques such as FFT (Fast Fourier Transform) and CFAR (Constant False Alarm Rate) [11]. Still, a number of processing steps can be toggled or adjusted by the

user, providing considerable flexibility. Processed data are stored in radar cubes – a common structure in radar applications [11]; which facilitates the implementation of more advanced post-processing algorithms. For visualization, the system supports both 2D and 3D representations.

The thesis is organized into five main chapters. The first provides the theoretical background of FMCW radar technology and its advantages over alternative radar methods. The second chapter focuses on the SiRad Easy[®] evaluation board, particularly its suitability for surveillance radar applications. The third chapter describes the design requirements and development process of the custom rotary platform, including its control software. The fourth chapter gives an overview of the MATLAB desktop application used to control the system and process radar data. Finally, the fifth chapter delves deeper into the data processing pipeline and available visualization methods.

1. FMCW Radar Fundamentals

Classical CW (Continuous Wave) radars broadcast a signal only on a single frequency, while FMCW (Frequency-Modulated Continuous Wave) 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 [12] 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

$$d = \frac{c_o \cdot t}{2}. \quad (1.1)$$

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

$$v = \frac{f_{\text{dop}} c_o}{2 f_{\text{rad}}}, \quad (1.2)$$

where f_{dop} is the Doppler frequency, c_o is the speed of light and f_{rad} is the frequency of the radar signal [6].

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 [6].

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

1. FMCW Radar Fundamentals

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 [6]. SiRad Easy[©] kit, used in this thesis, technically uses a segmented linear frequency modulation due to its limited computational power [13], but the mathematical principles are similar to those for sawtooth.

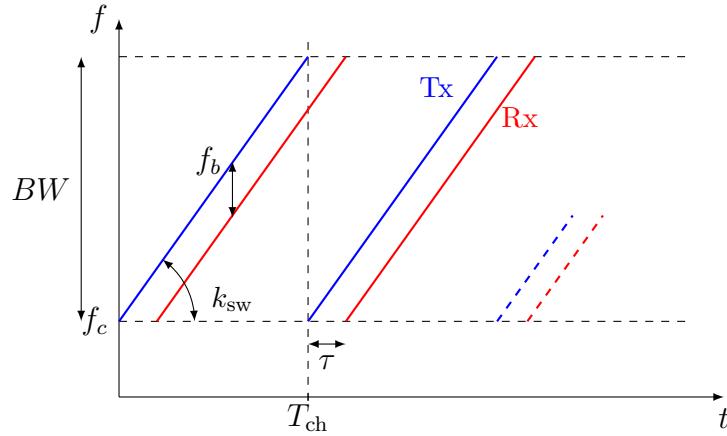


Figure 1.1: Ideal relation of frequency on time for received and sent signal [14]

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

$$\Delta f_s(t) = k_{\text{sw}}t = \frac{BW}{T_{\text{ch}}}t, \quad (1.3)$$

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

$$s_t(t) = A \cos \left(\omega_c t + 2\pi \int_0^t f_s(t) dt \right), \quad (1.4)$$

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 [15]

$$s_t(t) = A \cos(\omega_c t + \pi k_{\text{sw}} t^2). \quad (1.5)$$

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

$$s_r(t) = A \cos(\omega_c(t - \tau) + \pi k_{\text{sw}}(t - \tau)^2). \quad (1.6)$$

1. FMCW Radar Fundamentals

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

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

according to [17]. 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

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

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 [6].

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 [11].

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 [15]. 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

1. FMCW Radar Fundamentals

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 [6].

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 [18].

2. SiRad Easy[®]

Indie Semiconductor's SiRad Easy[®] is a 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 transceivers with two antennas: one for receiving and another for transmitting. So azimuth estimation is not possible out of the box.

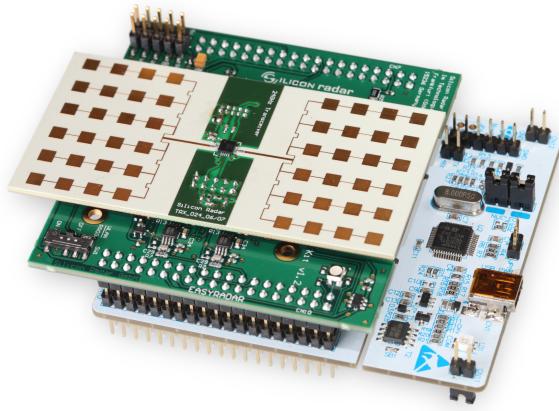


Figure 2.1: SiRad Easy[®] 24 GHz configuration [8]

Direct communication with the radar header is not possible, or at least, the communication interface is undocumented. Instead, there is always an intermediary in the form of an STM32 Nucleo series microcontroller. To this microcontroller, the user connects either directly with UART over USB or with WiFi, enabled by an on-board ESP32. Both are relatively low-bandwidth communication methods, with the serial maxing out at a baud rate of 1,000,000 [7].

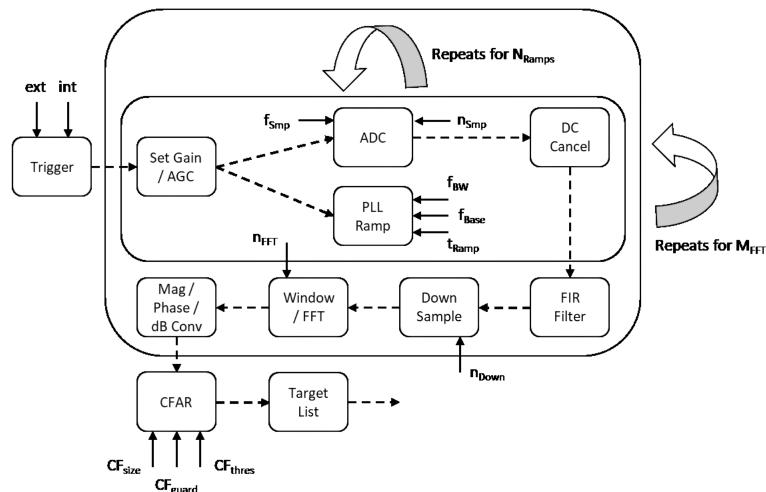


Figure 2.2: Flow of Radar Measurement on SiRad Easy[®] [13]

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

2. SiRad Easy[®]

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

This entire sequence started either by internal trigger or with external input (Be it via a GPIO pin or UART.). Once the trigger is received, the device carries out a user-configured number of chirps. Each with length determined by the number of sampling steps and the ADC sampling time as follows

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

The manufacturer recommends keeping r_{ramp} around 1 ms for a good SNR [13].

SiRad is designed to leverage coherent averaging to improve SNR of the output. This is achieved by broadcasting multiple ramps in series and averaging the results. While this technique was found to aid in measurement in a cluttered environments averaging effectively destroys the information about the Doppler shift. Additionally, due to the devkit's limited computational resources instead of a sawtooth waveform the radar broadcasts a segmented one with significant delays in between [13].

The devkit is clearly focused primarily on 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, broadcasted 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 that 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 timestamped with it's time of arrival which is assumed to be the time of chirp broadcast. Using this timing information speed spectrum can still be estimated using non-uniform DFT.

According to the radar's manual [13], the following equation was used to calculate the width of a single range bin:

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

where the variable N denotes the number of samples and $NFFT$ the 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 the user-chosen trigger period. However, as the minimal period of chirps is rather high, not to mention the radar doesn't

2. SiRad Easy[®]

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 and the resolution is only 0.031 ms^{-1} .

This lack of performance severely limits the radar's usability with regard to surveillance applications. Most surveillance radars are designed to monitor dynamic environments, where targets are moving, 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, in a parking lot, an operator can detect if a car is parked in a given spot or not. However, besides such binary information about the 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 more complex processing than if the spectrum was calculated on the radar itself. For these reasons, the default windowing and signal filtering were also disabled. The device was also set up to be triggered by a command sent over the serial line.

Another parameter to consider in relation to surveillance applications was AGC. Using it adds two additional ramps that are used solely to set the gain value, leading to some 3 ms delay. However, a bigger problem is the fact that the radar doesn't report the gain value. Using AGC could then lead to situations where the weighting of the data is not constant across rotation. 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 regard 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 and the number of samples, were left to the user to configure. Setting these strongly depends on which header is used and the environment in which the radar is in. Thus, making their configuration static wouldn't be a good idea.

2.2 24 GHz Header

The core block of 24 GHz header is a SISO TRX-024-007 transceiver, which integrates low noise amplifier, frequency mixer, filters and VCO into a single package. It is primarily designed to operate in the ISM band (24.0–24.25 GHz), but also supports an additional ultra-wideband mode with a frequency range

2. SiRad Easy[®]

from 23 GHz to 26 GHz [19]. On the SiRad Easy[®] there is no distinction made between those two modes and the user is free to set any bandwidth they want [13]. The transmitter output power ranges from 2.5 dBm to 6 dBm, depending on the configuration [19]. A maximal range of 400 m is advertised [8], though in practice only ranges of around 300 m were achieved, with target being a multistory building.

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 [19].

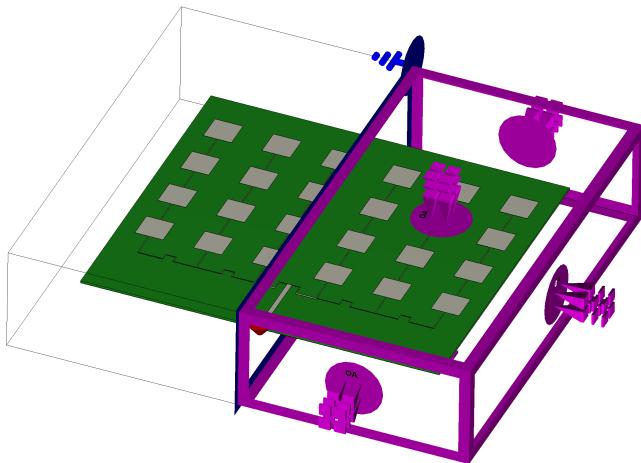


Figure 2.3: Simulated 24 GHz header with boundary conditions showed

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 a $H_{\text{norm}} = 0$ 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). The lack of a minimum at 24 GHz may be attributed to difficulties in 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.

2. SiRad Easy[®]

It's clear that, especially in the 90-degree norm, the beam isn't that narrow. Thus, this header is not suitable for applications where accurate elevation information is needed. However, if only a low degree of accuracy is required or a wider field of view would be beneficial, this header is a good choice. With wider beam the radar can gather more information about the environment faster compared to one with a narrower beam. In industry, some surveillance radars are also designed to have a wide beam in elevation, such as some Air Traffic Control radars [20] or even some ground radars [4]. Still, usually, solutions such as low/high beam switching are preferred instead of relying purely on a wide beam [21].

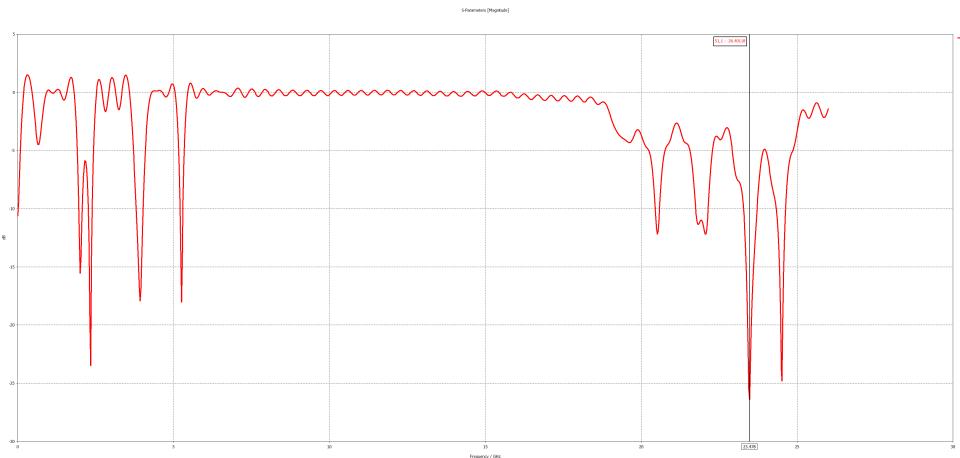


Figure 2.4: s_{11} parameter of the 24 GHz header

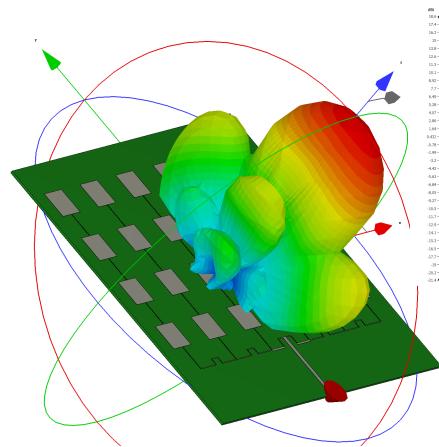
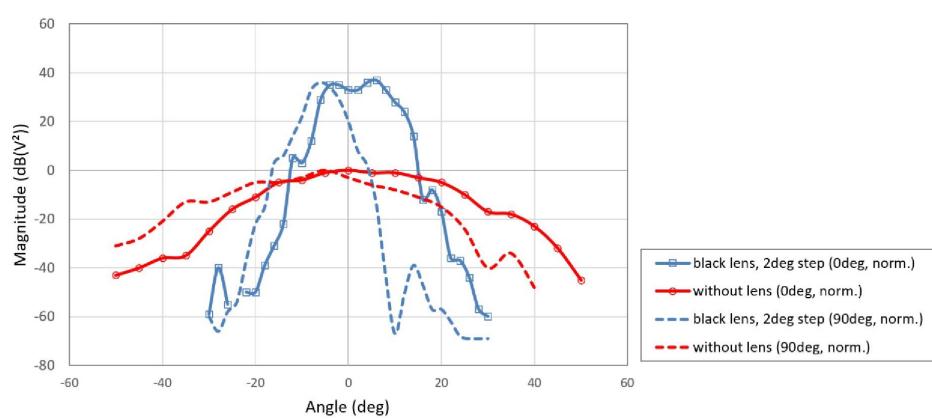
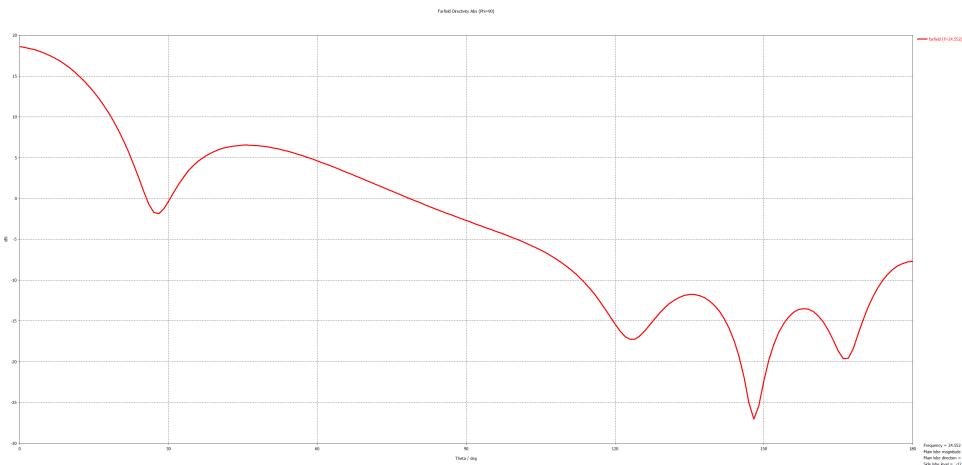
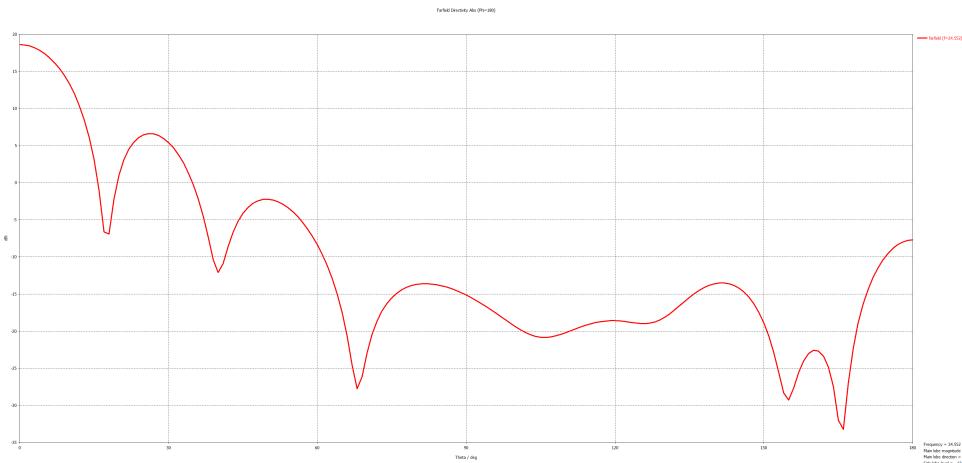


Figure 2.5: Radiation pattern of 24 GHz header – 3D view

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 an output power ranging from -7 dBm to 1 dBm [23]. The chip is advertised as capable of detecting large targets at distances of up to 40 m [8].

2. SiRad Easy[®]



Out of the box performance of the system is quite bad (Figure 2.8) with the width of the main lobe being roughly $\pm 40^\circ$ 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 $\pm 4^\circ$ [22] (Figure 2.8).

3. Rotary Platform

The following chapter outlines the design process and operational principles of a rotary platform specifically developed for the 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 classic contact slip ring that features a dedicated USB 2.0 connection along with 8 additional signal wires. Advertised insertion is less than 2 dB [24]. More problematic than the loss, however, is crosstalk between signal wires when the pitch control stepper motor is running. This necessitated the addition of a small capacitor to the endstop signal wire to filter out high-frequency noise.

While the USB 2.0 connection does not seem affected by the crosstalk, its connection still isn't ideal. Without a signal conditioner connected directly to the slip ring output, signal quality is poor enough that a direct connection to the PC is required – thus, the radar needs to be some 20 cm from the PC. Even then, depending on the 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 the user requires more granular control over the position, it can be facilitated by microstepping.

Similarly, high-speed movement was unnecessary for this application. As was already outlined, the radar is capable at best of some 50 Hz update rate. With such an update frequency, even at a low 8 RPM, we travel a degree in under 21 ms – any faster and we would not 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 slip ring.

The rest of the design is relatively simple. Slipring is mounted to a 3D printed base that elevates it slightly above ground. Stepper motor is then mounted directly under the slip ring, with a shaft connecting it to the 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 the 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 the platform, is attached to the second support strut.

Unfortunately, while the design offers some ability to tension the belt, in the currently printed configuration the adjustment range is not sufficient to fully tension the belt. While slippage is not a problem, there is some slack in the radar's tilt. However, as the radar's capabilities in the pitch direction are not

3. Rotary Platform

particularly high, it was decided not to reprint the whole assembly, but only to fix the 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, shaft, 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.

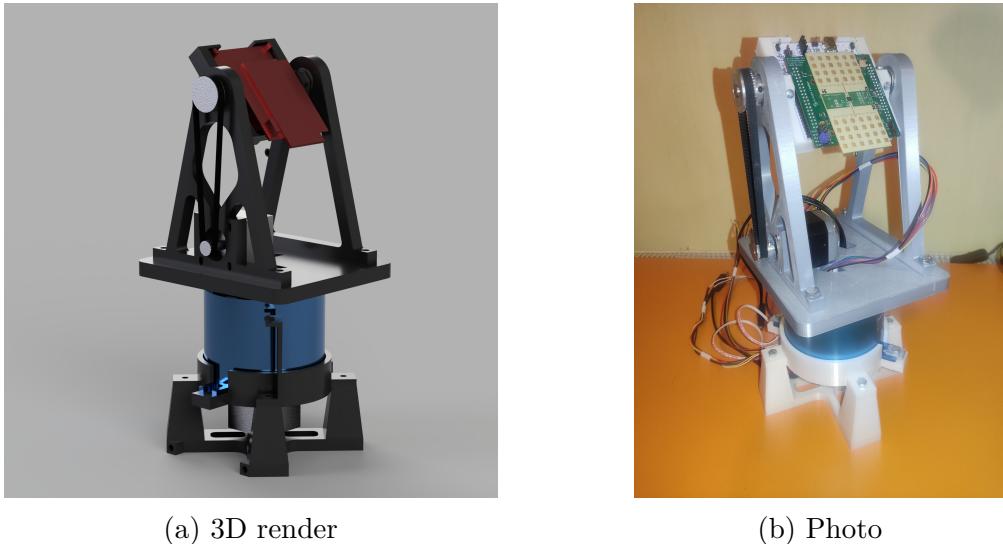


Figure 3.1: Form of the final assembly

3.1.1 Platform Electronics

The electronic side of the platform is relatively simple. Only two 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, up to 1:16 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 arises, the A4988 development kits can be replaced by ones with a different IC, such as the TCM2209, as manufacturers keep the same pinout and dimensions on these. This might be necessary in applications where platform loudness would be a serious problem, as the A4988 produces a rather noisy driving signal. More modern drivers exhibit up to 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 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

3. Rotary Platform

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 used 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 basic G-code interpreters, the platform should support additional features to reduce the user's manual control burden. These include common features such as defining movement limits, absolute/relative positioning modes, spindle mode and automatic homing. Lastly, the platform should support a programming interface that allows an application to send an entire choreography of movements to the platform at once, after which it only needs to monitor its status.

For uplink communication, the platform must provide real-time information about its current position. This data then enables the user to make necessary mathematical corrections and properly interpret the 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 and command parsing, and the other manages execution—was deemed unsuitable for this use case. Such an approach would complicate the integration of a 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 the transition from one command to the next is nearly instantaneous.

It is necessary to note that the platform does not store any permanent configuration, including microstepping settings. This approach was taken because the platform does not offer any easy way to display the current settings and is expected to be operated only from a computer application. Thus, instead of needing to keep track of which settings are on the PC and which are stored on the platform, a simpler approach was taken.

3.2.1 Communication Layer

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

Immediately after parsing, a response is sent to the user confirming whether the command was parsed correctly or not. However, as the communication layer

3. Rotary Platform

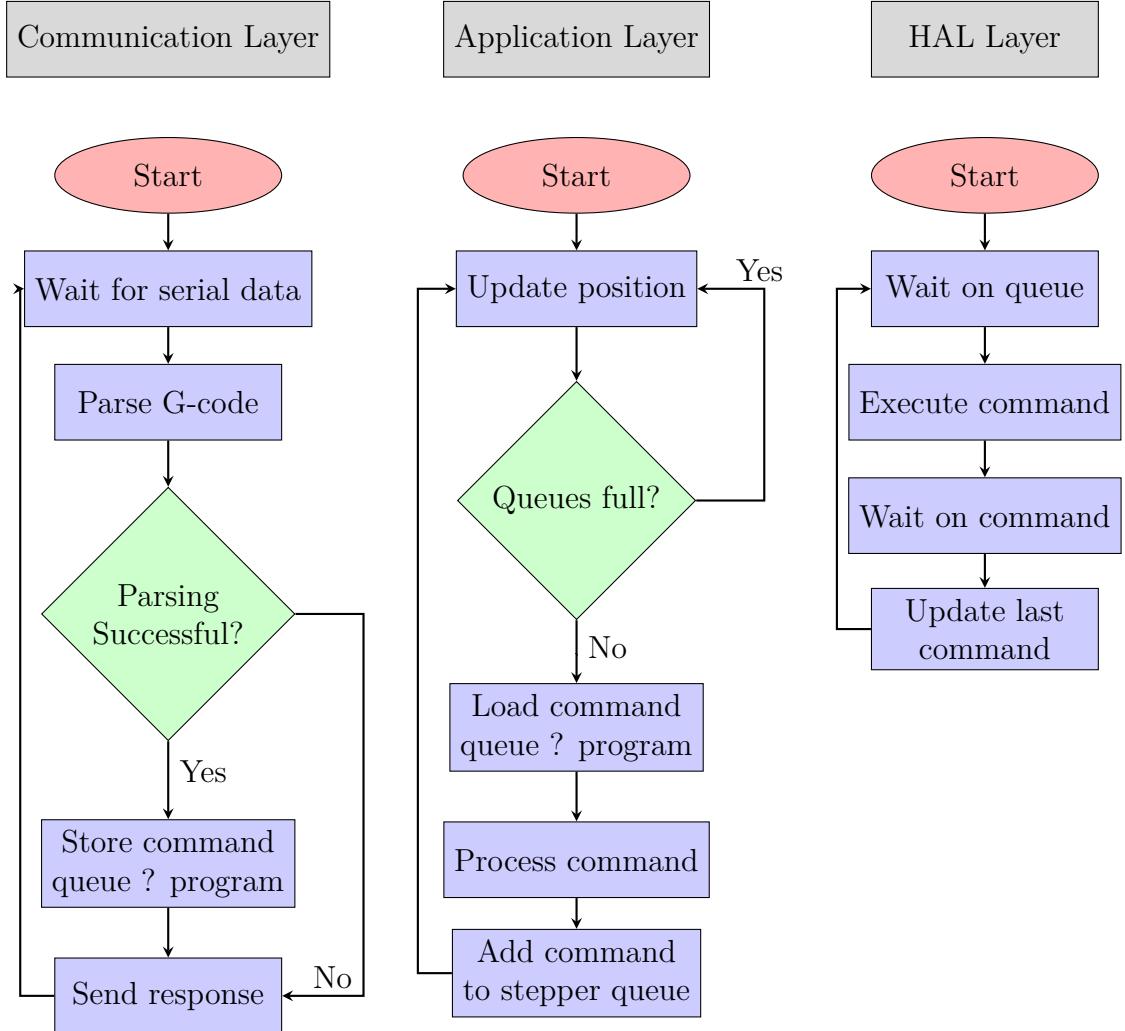


Figure 3.2: Program flow diagram

does not and cannot check the command within the context of all previous commands, it is possible that a 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 the current device position and preparing commands to be sent to the stepper motors. Aside from the 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.

A key departure from standard G-code interpreters, such as [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 the 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.

3. Rotary Platform

This behavior is a necessary side effect of the spindle regime, which typically can only be toggled on or off dynamically [28]. 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 compute absolute positioning commands, thus necessitating relative positioning. However, it would be rather restrictive to force the user to use relative positioning on the second axis; therefore, the positioning modes are independent.

To support other possible applications, a manual override mode was also implemented. This enables the user to manually push a move command directly to the stepper queues, totally skipping the application layer. The primary use case for this mode is to allow tracking of targets or use in other applications that require real-time control over the platform. However, user must keep in mind that 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 calculations are handled in the application layer, the whole routine remains highly efficient – the layer just moves stepper a given number of steps, or let's it rotate in case of spindle mode.

The main challenge was the generation of precise PWM signals (used to control stepper motor drivers) and stopping the signal generation after a specific number of steps. As generating a PWM signal in software is impractical, dedicated ESP32C6 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: for example, 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, combination of MCPWM and PCNT was chosen. MCPWM handles pulse generation, while PCNT counts steps, enabling easy control over movement and providing a 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 at higher speeds. Nevertheless, when measuring the time for 24,000 steps

3. Rotary Platform

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

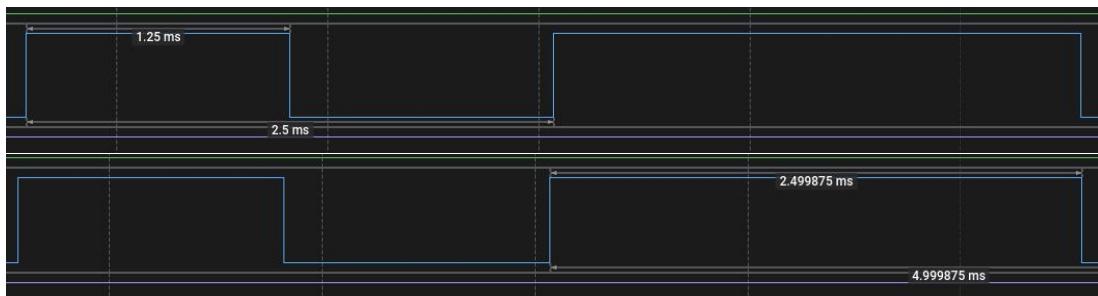


Figure 3.3: Moment of change between commands (120RPM \Rightarrow 60RPM)

An attempt was made to also measure the delay between switching commands, as displayed in figure 3.3. The results indicate that the delay between commands is 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 the stepper queues are supplied with commands in advance, the platform can operate without noticeable interruptions.

4. Control Application

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

4.1 Main Window

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

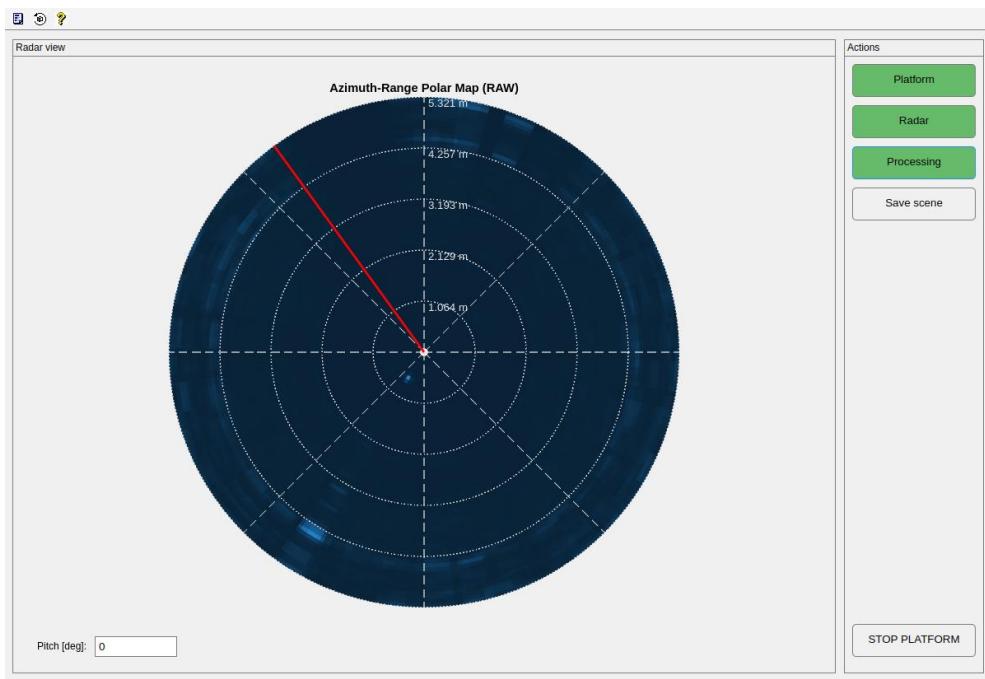


Figure 4.1: Main application window

4.1.1 Application Configuration

As this thesis concerns itself with a rather generic implementation of a surveillance radar system, the application is designed to be as flexible as possible. For this purpose, a 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 the extensive configuration options, the user is expected to have some knowledge about the subject to achieve satisfactory results. Firstly, configuration options are not checked against each other to verify if they are mutually exclusive. For example, for speed calculation, ramp averaging must be turned off, but as things stand, the application will allow

4. Control Application

the user to continue even with it enabled. Secondly, configuration of components such as CFAR or DBSCAN strongly depends on bandwidth, number of samples, FFT point count, and other parameters. A CFAR guard parameter of 3 might be too small on a 5 GHz bandwidth with a single range bin width of 50 mm, but with $BW = 500$ MHz it could be way too large. To aid in choosing correct parameters, the application provides some basic information, like the dimensions of the bins. Still, 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. The first configures serial port connections to the radar and the platform. The second section handles configuration of the platform, followed by a section designated for radar configuration. The last section configures how the data are processed and via which method they are visualized.

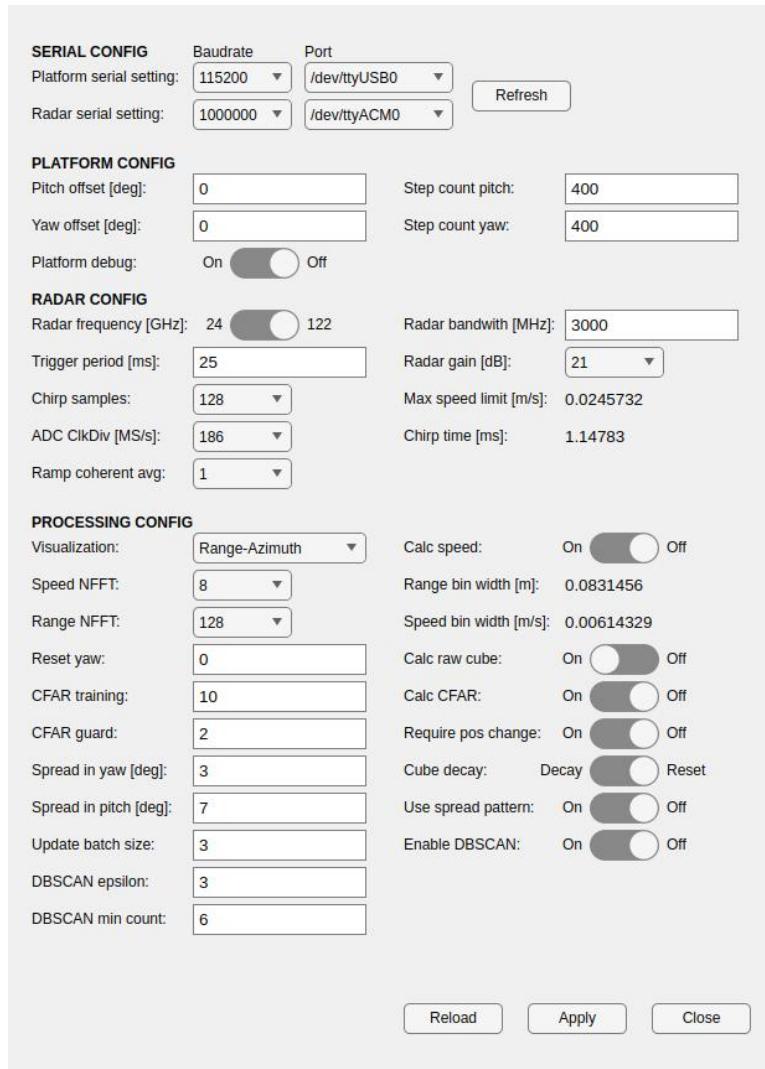


Figure 4.2: Preferences menu

User preferences are stored in an INI file in a directory whose path depends on the operating system used—`%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. Control Application

4.2 Platform Control Interface

It is necessary to emphasize that the platform control is entirely independent of radar operation. The user loads a program onto the platform and starts its execution. From then on, the application only processes current position reports coming 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 which movements comprise the platform's current program.

In the preferences menu, the user can configure static offsets to the platform with the **Pitch offset** and **Yaw offset** options. This is useful in case the zero position of the platform is not aligned with what the user wants to be zero in the data. Aside from offset, step count configuration is also available here with **Step count pitch** and **Step count yaw**. Step count takes into account both microstepping and the physical down-gearing ratio. It is important to note that, as the platform doesn't store the step count in non-volatile memory, this setting is essential to maintain proper operation of the platform. The step count command is sent to the platform each time it is connected or the configuration is changed. Lastly, the **Platform debug** toggle enables full debug output to be visible in the 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 the user to load, edit, and send programs to the platform. The left sidebar allows the user to load stored programs by clicking on their name. The middle text boxes are used to edit the program header (top part) and program body (middle part), with the bottom text field being used to display the platform's debug output. On the right side are buttons that handle basic operations on the 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 the start of a program, end of

4. Control Application

header, and so on are automatically sent to the device when the rest of the program is uploaded. To validate whether the program was parsed correctly, the user can check the text window at the bottom of the screen, which forwards the platform’s debug output. If the user clicks the store button, the program is stored in the same INI file as the preferences.

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

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

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 the preferences menu, there are some parameters that can be configured.

The `Radar frequency` toggle chooses between the 24 GHz and 122 GHz headers. However, this only affects the configuration sent to the radar; it does not reflash the firmware, which is also necessary. After the base frequency, `Radar bandwidth` can also be configured; both positive and negative values are accepted, however manual verification is required to ensure that the radar is working as expected. While the manufacturer states a 3 GHz limit on the 24 GHz header and 5 GHz on the 122 GHz header [13], in practice—especially with the 24 GHz header—achievable bandwidth was usually much lower and its limit was rather inconsistent.

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

The drop-down menu `Ramp coherent avg` allows the user to pick how many ramps will be averaged together. However, use of this feature is generally not recommended unless the surrounding environment is too cluttered. Each ramp adds about 2 ms overhead to the radar’s operation and makes speed calculation impossible.

The last parameter is `Trigger period`, which dictates how often the application will send a trigger command to the radar. As previously stated, the radar is at best capable of a chirp roughly every 20 ms.

5. Radar Data Processing

Firstly, it is necessary to outline the inputs and outputs of the entire processing pipeline. Data are arrive to the application from two sources: serial connections to the radar and to the platform. These two data streams are then paired together based on timestamps logged when the data are received. Program then performs user-selected calculations – components such as CFAR calculation, speed calculation, and so on can be enabled or disabled as needed. After processing finishes, the output data are stored in a radar cube, a common method for managing radar data that enables easy integration of more complex algorithms in the future [11].

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

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

The main complicating factor in the processing was the need to keep the main thread responsive. The most time-critical responsibility of the main thread is the timely readout of data from the serial ports. If data cannot be read immediately upon arrival, the assigned timestamps will be incorrect and, therefore, the data will be attributed to a wrong position. In addition, the GUI becomes quite unresponsive if the main thread's resources are stretched too thin.

To solve this problem, processing heavily leverages MATLAB's Parallel Computing Toolbox, offloading as much as possible to different cores. Throughout the entire processing pipeline, there are mechanisms for buffering data into larger batches, and if needed, discarding some in case the software is unable to keep up. Unfortunately, as MATLAB's parallelization capabilities with threads are rather limited – external language interfaces do not work, nor do certain file or memory mapping functions [32] – the parallelization needed to be done with processes. This restricts 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.

The processing pipeline can be divided into several distinct steps as follows:

1. Data acquisition: Data are read, a simple FFT is calculated, and spectra are buffered into a circular buffer.
2. Data processing: Spectra are retrieved from the circular buffer, paired with positions, and processed.
3. Cube update: Processed data are buffered into a ping-pong buffer. If the buffer is full, the cube update routine is launched.

5. Radar Data Processing

4. Visualization: After cubes are updated, data are visualized.

5.1 Data Acquisition

The basic workflow of this step is shown in figure 5.1. The process starts as soon as MATLAB detects a line terminator within the incoming radar data stream. After the entire frame from the radar is read, the data are timestamped and the I and Q channels are buffered. The observer `dataProcessor.m` is then notified and left to retrieve the data when the main thread is free.

After retrieval, the in-phase and quadrature channels are merged to form a complex signal, which is multiplied by a Hann window to prevent spectral leakage. A basic 1D FFT is then calculated, with the number of points set by the `Range NFFT` parameter in the preferences. If the FFT order is larger than the number of samples, zero padding is added. The resulting spectrum is again buffered in its complex form. Following this step, the program has in memory multiple FFT spectra – these can be leveraged to enable range-Doppler map computation.

After the spectrum is stored, the program verifies if the platform has moved. This check helps save processing resources by avoiding computation of an R-D map that would be immediately overwritten. After movement is detected, the code then takes all spectra (Except the last one when the platform has changed position.) and launches another processing step in a separate thread. If the user wishes processing to always take place (e.g., when verifying radar settings while the platform is static), this check can be skipped by toggling `Require pos change`.

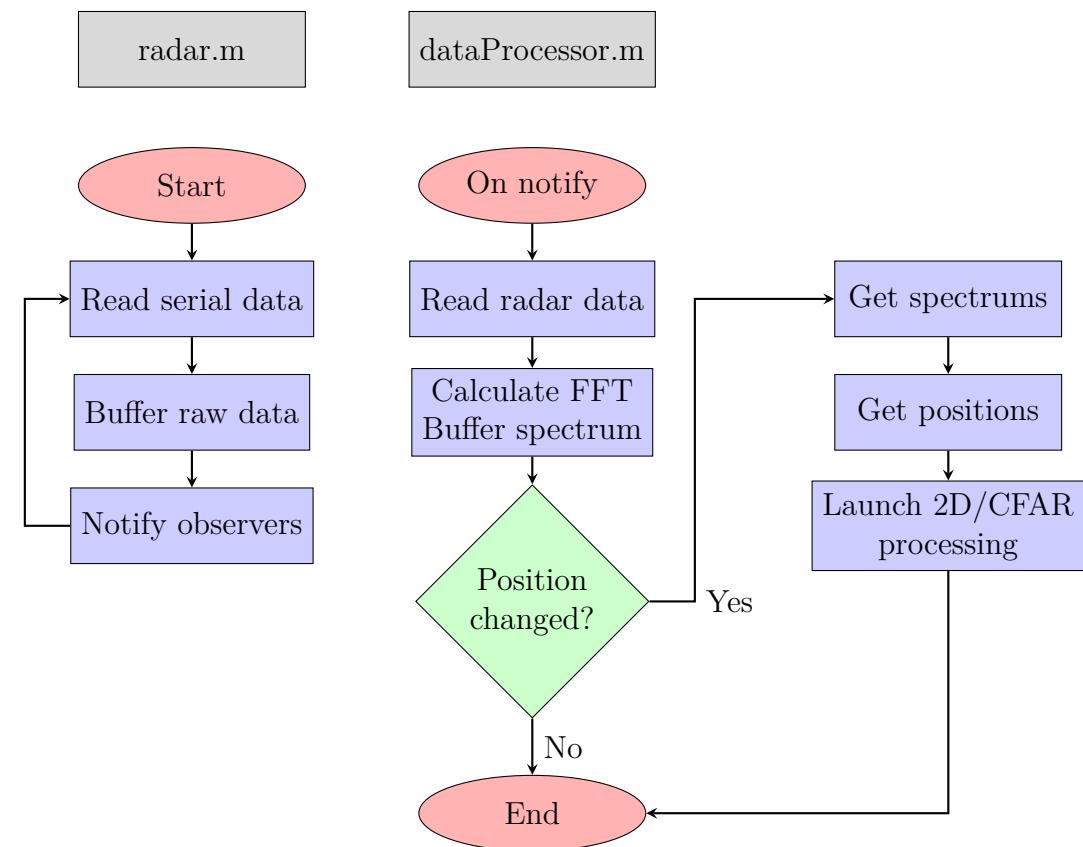


Figure 5.1: Processing flow - Data acquisition

5.2 Data Processing

A parallel process takes calculated FFT spectra and their accompanying positions and processes them into a single range-Doppler map and an array with CFAR output. A basic outline of this step is shown in figure 5.2.

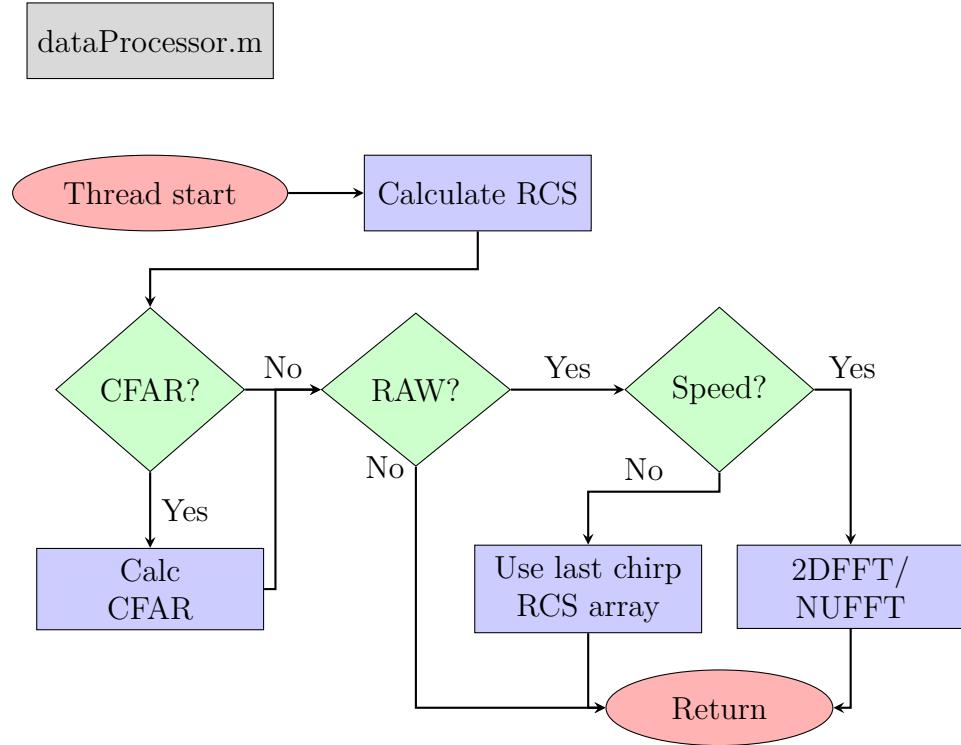


Figure 5.2: Processing flow - Data processing

In the first step, the 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 the fourth power of the distance to the target. Range can be calculated from the manufacturer-supplied equation (2.2), and power is proportional to the square of values reported from the radar. Given that the constants of the radar equation change based on configuration, calibration would be required in order to obtain real RCS values. However, this calibration is not supported, and thus onward, when referring to RCS we refer to values proportional to the RCS, not the actual RCS of the object.

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

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

5. Radar Data Processing

When speed calculation is enabled, the program first cuts off samples that are too far away from the position of the latest chirp in the batch. After that, the timing of chirps is analyzed to determine whether NUFFT needs to be used. The decision about which transformation to use is based solely on the ratio of the 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, a more complex method should be used, but as the radar timing data are not very precise – not to mention the radar can track only very slow-moving objects – this method was deemed sufficient. For similar reasons, no correction for the platform’s movement is applied. The output range-Doppler map is then shifted in order to create a 2D array with Range NFFT \times Speed NFFT elements, with speed ranging from $-\frac{\text{Speed NFFT}}{2} \cdot w$ to $(\frac{\text{Speed NFFT}}{2} - 1) \cdot w$, where w is the speed bin width calculated with (1.18).

5.3 Cube Update

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

As MATLAB processes do not share the same memory space by default, to avoid redundant copies of the cube, MATLAB’s memory mapping is used. Using the `memmapfile` function, the cube is mapped to a file on a 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 testing showed that for cubes of size 512 MB (roughly a cube for a range-Doppler map of 256 \times 16), the program was still able to keep up with the radar.

The update routine for cubes storing CFAR and raw data is quite similar in the case the spread pattern is disabled. New data are simply added to the cube at their respective positions, replacing the old data. If the cube is set to decay (The `Cube decay` option is set to ‘decay’.), the entire cube is multiplied by a decay factor derived from the platform’s movement speed. The slower the platform moves, the slower the data decay. Using this option leads to an output that more visually resembles traditional analogue radar systems, where the most recent data are brighter than the older ones. However, the user needs to keep in mind that color changes do not 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 need to be buffered.

Even on smaller cubes of around 256 MB, this operation takes in the worst case about 50 ms to complete on a laptop with an i7-10210U CPU. If this behaviour would be undesirable, decay can be turned off in the settings in favor of a hard reset of the cube based on a trigger at a specified yaw angle. The rest of the processing is then fast enough to keep up with the data stream without any buffering.

Enabling the spread pattern leads to degradation of the quality of the data in exchange for better visualization quality – especially in range-azimuth visualization style. Instead of adding new data to just one position in the cube, the

5. Radar Data Processing

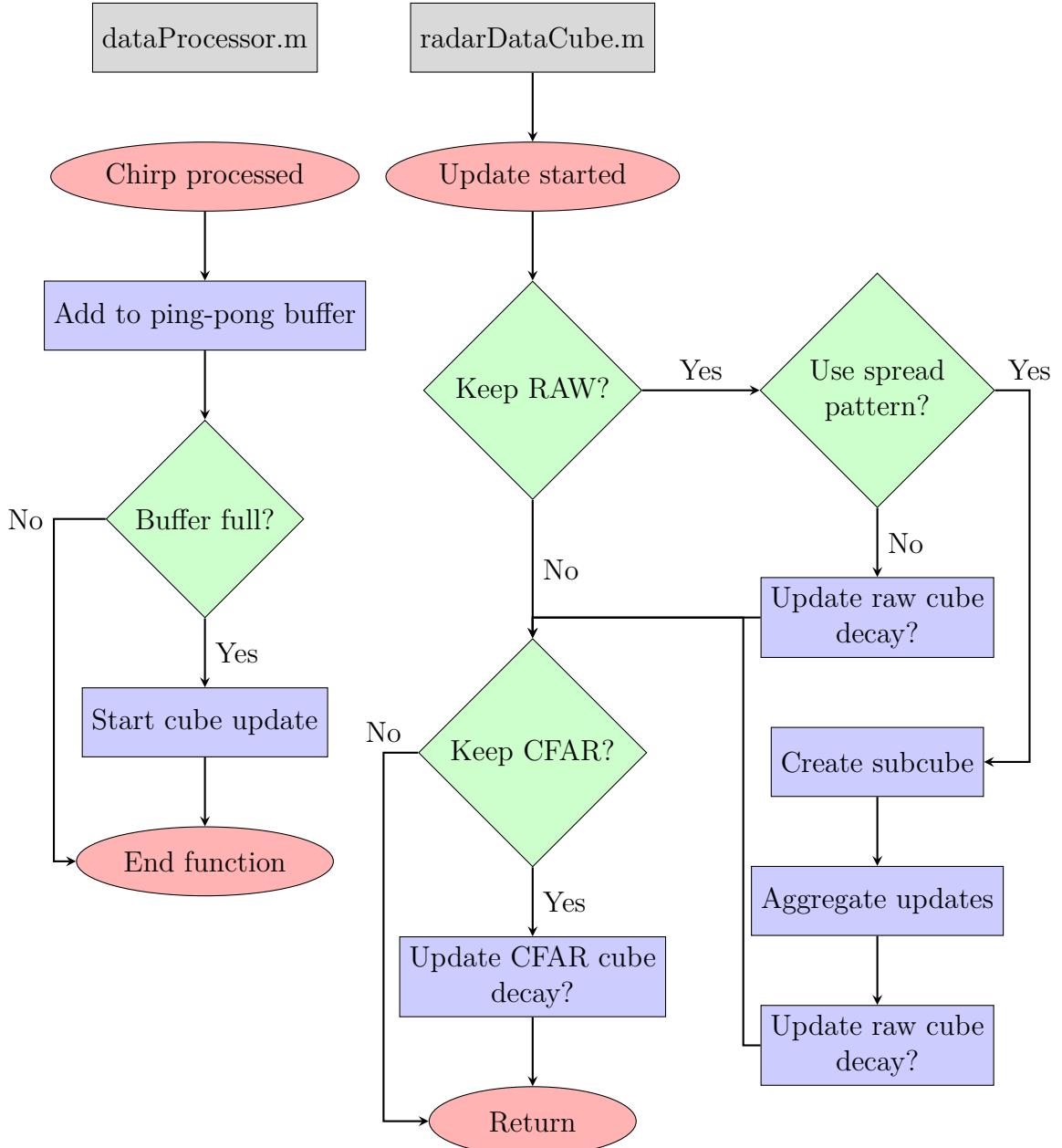


Figure 5.3: Processing flow - Cube update

range-Doppler map is spread over a larger area with the 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, the update process can be made more efficient. Instead of directly adding these contributions to the full cube, a much smaller cube is created. That cube 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 the smaller memory size enables better caching. Still, in the case of a more performant radar system, this method would be unnecessary, not to mention it has very little basis in radar data processing theory.

Due to the limited performance of MATLAB on the 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 the AVX2 x86 instruction set extension to speed up floating point operations. Chiefly, decaying of the cube

5. Radar Data Processing

exhibited roughly a twofold performance increase thanks to SIMD instructions.

5.4 Visualization

After the cube update finishes, the program returns to the main thread and the data are visualized. The only way the user can see the data is via visualization. Text output or access via API is 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 the x-axis and Doppler on the y-axis; if speed is not calculated, the y-axis is replaced with the 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 a polar plot for a fixed pitch angle, which is chosen by a text box at the bottom of the screen. A demonstration of a Range-Azimuth map can be seen in figure 4.1, which captures testing with the 122 GHz header and a rather large bandwidth of 3 GHz, providing only about 5 meters of range. In the picture, only raw data are displayed; however, this visualization style is also able to display CFAR data or overlay both.

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

This visualization style might be especially useful when paired with the 24 GHz header. Due to its rather wide radiation pattern, the radar is able to detect objects in quite a large area. This can be extended even further if the spread pattern is enabled, where more pitch angles will influence the data. Thus, for a realization of a classical surveillance radar, this method of visualization might be the 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 the Range-Azimuth map, both angles are configurable via text boxes at the bottom of the screen. In cases when speed calculation is disabled, the y-axis is replaced with the RCS value.

The main purpose of this visualization style is to provide a method for verifying radar data processing and settings. As the devkit does not report many 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 as expected.

5. Radar Data Processing

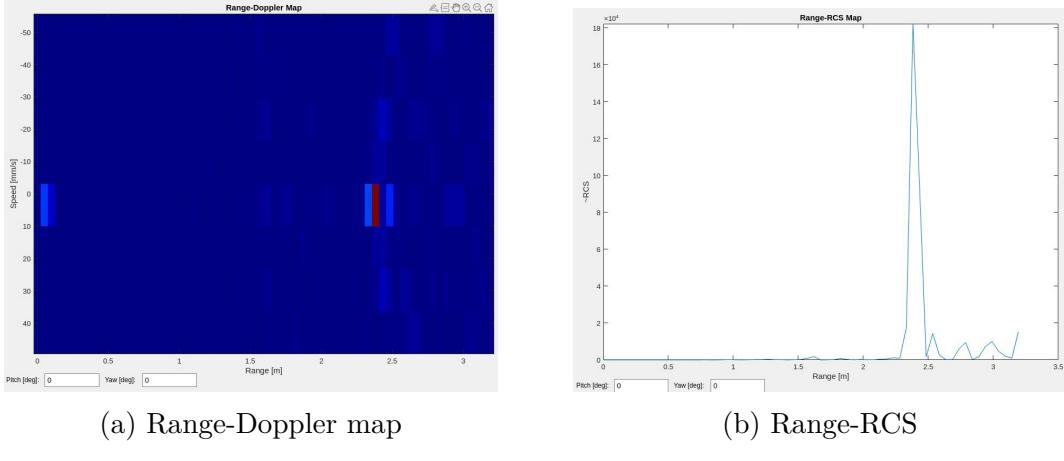


Figure 5.4: Range-Doppler and Range-RCS maps

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

5.4.3 Target-3D

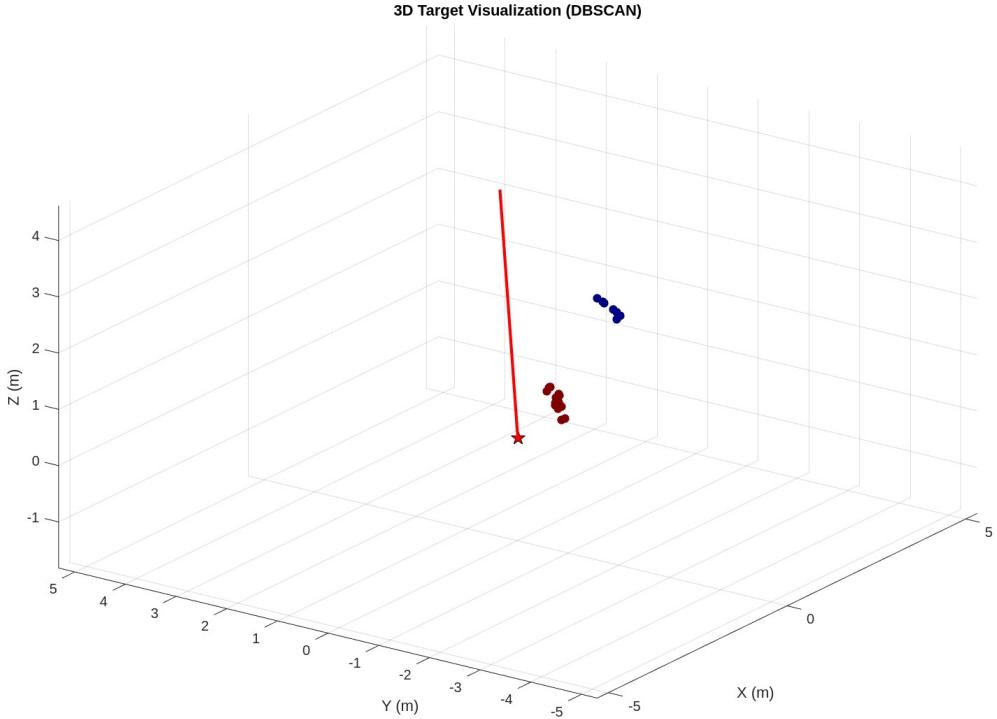


Figure 5.5: 3D Space visualization method

This visualization style is used to display CFAR data in 3D Cartesian space (Figure 5.5). The 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. If decay is enabled, then a small threshold is used to remove points whose values have sufficiently degraded. To aid with orientation within the plot, the current heading of the platform is displayed as a red line.

5. Radar Data Processing

If the environment is too cluttered and CFAR generates too many points, the user can enable basic clustering with MATLAB's built-in DBSCAN function [35]. The general idea of DBSCAN is that, for a given point in the dataset, distances to other points are calculated and compared with a user-defined threshold ε . If a sufficient number N of points is found within the threshold, the point is marked as part of a cluster [36]. In MATLAB's implementation, the distance threshold and minimum number of points are both configurable variables, with the other input being just the list of points [35]. As the original data are in a polar coordinate system, a problem arises with calculating distances given that, in such a system, the 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.

These problems can be overcome using many different methods, such as using adaptive thresholds in relation to radius [36] or relying on more clever axis systems than just polar or Cartesian [37]. However, as in this thesis 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 first converted to Cartesian coordinates, then distance is calculated using the standard Euclidean formula. This distance is then divided by the average radii of the points. It needs to be said that, while dividing by the mean of distances is sometimes used when working with spherical data [38], the author is not aware of any literature that would suggest its use in DBSCAN. Especially at longer distances, this method will lead to significant errors.

Conclusion

The goal of this thesis was to develop a surveillance radar system based on FMCW technology. This approach enables accurate distance measurements of targets with relatively low power consumption. Rather than adopting a more conventional MIMO system, a simpler solution was chosen: a single RX and TX antenna, with mechanical beam steering provided by a custom rotary platform.

Using off-the-shelf components and 3D-printed parts, a two-axis rotary platform was designed and constructed. Despite minor issues with belt tension on the pitch axis, the platform reliably controls the radar's position in both yaw and pitch, with a standard accuracy of 1.8° , further improvable via microstepping. Additional features such as automatic homing and rotation limits were implemented for improved usability. Thanks to its similarity to other G-code-based systems, the platform is readily adaptable to other applications. Moreover, the platform was designed with capabilities beyond the requirements of this thesis, and during testing it executed commands accurately and without noticeable delay.

The capabilities of the SiRad Easy[©] evaluation board were analyzed and found to be rather limited for use in surveillance radar. Its low and inconsistent reporting rate (50 Hz) restricts the maximum detectable speed well below one meter per second. This limitation effectively prevents the tracking of moving targets – a common use case for surveillance radars – and thus significantly reduced the scope of this thesis. The system is therefore mainly suitable for static scenes, providing binary detection of target presence. Nevertheless, within this limitation, the radar still provides good resolution across a variety of ranges, thanks to the ability to switch between the 24GHz and 122 GHz headers. Both close range application with high resolution or long range (up to some 300 meters) are possible.

A MATLAB-based control application was developed to integrate both rotary platform management and radar data processing in a single package. The data processing pipeline follows standard approaches, employing techniques such as FFT, CFAR, and DBSCAN, and stores results in radar cubes.. Extensive parallelization was used to ensure timely processing, even for computationally intensive tasks like decaying the entire cube. 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.

Due to the generic design requirements, the processing pipeline is highly customizable. Settings such as the number of FFT points, step count per rotation, and CFAR parameters can all be tailored for specific applications. However, this flexibility requires users to have a solid understanding of the underlying algorithms and principles.

Only visual output was implemented for this thesis, with both 2D and 3D visualization options available. The classical 2D Range-Azimuth map is likely the most useful, providing information about a significant portion of the environment, given the radar's radiation pattern. In case of 3D visualization only CFAR data are visualized in exchange for a more detailed view of the environment by providing pitch information. If deployed in a cluttered environment, the DBSCAN algorithm can be used to filter out unwanted noise, but in this thesis, its

implementation is mainly illustrative and not thoroughly optimized or correctly configured.

Importantly, the processing pipeline was designed such that, if the radar module were replaced with a faster unit, the system's capabilities could be extended without significant changes to the codebase. More complex operations, such as 2D FFT or cube updates, are executed in parallel and are triggered by platform movement rather than each radar update. Therefore, increasing the number of chirps would not significantly impact performance.

However, if the number of FFT points (for speed or range) were to increase substantially, beyond what was tested (up to about 500 MB cubes), a different cube update strategy would be needed. One possibility would be to leverage GPU acceleration for cube operations. Alternatively, with only a CPU, the cube could be split into smaller chunks, loading only the relevant portions into RAM according to the platform's position and direction. However, the decay operation would be nearly impossible to realize quickly in case of large cubes without GPU acceleration.

In conclusion, a basic surveillance radar system was successfully designed and implemented. It is capable of detecting targets over a wide area, with a basic resolution of 1 degree in yaw and pitch, and range accuracy from centimeters to decimeters. The processing pipeline offers extensive customization, allowing users to fine-tune the system to their needs. However, due to the limitations of the radar module, the system is not suitable for applications requiring the 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](https://doi.org/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](https://doi.org/10.3390/s130404527).
- [4] *Revolution 360 Ground Surveillance Radar* [online]. [visited on 2025-05-10]. Available from: <https://www.blighter.com/products/revolution-360-ground-surveillance-radar/>.
- [5] BROOKER, Graham. Long-Range Imaging Radar for Autonomous Navigation. 2005. Available also from: <https://ses.library.usyd.edu.au/handle/2123/658>.
- [6] JANKIRAMAN, Mohinder. *FMCW radar design*. Boston: Artech House, [2018]. ISBN 978-1-63081-567-7.
- [7] 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.ffa.indiesemi.com/datasheets/User_Guide_Easy_Simple_V2.5.pdf.
- [8] INDIE SEMICONDUCTORS GMBH. *SiRad Easy® r4, SiRad Easy® & SiRad Simple®: User Guide*. 2017. Version 2.1.
- [9] *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.
- [10] *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>.
- [11] RICHARDS, Mark A. *Fundamentals of radar signal processing*. Third edition. New York: McGraw Hill, [2022]. ISBN 978-1-260-46871-7.
- [12] *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>.
- [13] 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.ffa.indiesemi.com/datasheets/Protocol_Description_Easy_Simple_V2.1.pdf.

-
- [14] 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_Radar.pdf.
 - [15] SULEYMANOV, Suleyman. Design and Implementation of an FMCW Radar Signal Processing Module for Automotive Applications. 2016. Available also from: <http://essay.utwente.nl/70986/>.
 - [16] 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.
 - [17] 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/>.
 - [18] 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.
 - [19] 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.
 - [20] *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>.
 - [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

1.1	Ideal relation of frequency on time for received and sent signal [14]	5
2.1	SiRad Easy [®] 24 GHz configuration [8]	10
2.2	Flow of Radar Measurement on SiRad Easy [®] [13]	10
2.3	Simulated 24 GHz header with boundary conditions showed	13
2.4	s_{11} parameter of the 24 GHz header	14
2.5	Radiation pattern of 24 GHz header – 3D view	14
2.6	Radiation pattern of 24 GHz header – 180° norm	15
2.7	Radiation pattern of 24 GHz header – 90° norm	15
2.8	Radiation pattern of 122 GHz header comparison [22]	15
3.1	Form of the final assembly	17
3.2	Program flow diagram	19
3.3	Moment of change between commands with 120RPM and 60RPM	21
4.1	Main application window	22
4.2	Preferences menu	23
4.3	GUI of the platform control	24
5.1	Processing flow - Data acquisition	27
5.2	Processing flow - Data processing	28
5.3	Processing flow - Cube update	30
5.4	Range-Doppler and Range-RCS maps	32
5.5	3D Space visualization method	32

List of Tables

3.1 Stability of PWM generation	21
---	----

Appendix

Appendix A – G-code glossary

Following sections act as a glossary of all G-code commands used in the platform control. More in-depth descriptions of the commands can be found in the source code documentation.

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 $[0, 360^\circ]$ or $[0, \text{STEP_COUNT}]$
 - 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:** Continuous 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 and one in the other
 - `low < high`: Valid range $[\text{low}, \text{high}]$
 - `low > high`: Valid range $[\text{low}, 360^\circ] \cup [0, \text{high}]$
 - Out-of-range targets snap to nearest valid position
- **Programming:** G-code program can be uploaded to the device and executed
- **Synchronization:** If a command is issued to both axes, they will wait for each other; otherwise, the other axis will be free to execute the 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 the speed of yaw and pitch respectively. In the case of spindle, Y or P are not followed by step count/angle but instead by 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 an M03 command to a stepper in absolute positioning mode, it will be automatically switched to relative positioning and throw an error
- If no argument is provided, both axes 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