Indonesian Journal of Aerospace Vol. 21 No. 2 December 2023 : pp 83 – 98 (Wiyono et al.)
DOI: 10.55981/ijoa.2023.2516
https://ejournal.brin.go.id/ijoa/

Implementation of Back-projection Algorithm for Synthetic Aperture Radar Image Processing on Low-Cost Hardware Platform

Agus Wiyono[1], Nurul Chasanah[1,2], Jefri Abner[1], Abdurrasyid Ruhiyat[1,2], Abdul Rohman[1], Farohaji Kurniawan[1], Muksin[1], Satria Arief Aditya[1], Agus Hendra Wahyudi[1], Novelita Rahayu[1], Andria Arisal[3], Bambang Setiadi[4]

[1]Research Center of Aeronautics Technology, BRIN, Indonesia
[2]Department of Electrical Engineering, Universitas Indonesia, Indonesia
[3]Research Center of Data and Information Science, BRIN, Indonesia
[4]Research Center of Telecommunication, BRIN, Indonesia
e-mail: agus115@brin.go.id

Received: 09-11-2023 Accepted: 29-11-2023 Publishied: 31-12-2023

Abstract

This work presents the implementation of back-projection algorithm for Synthetic-Aperture Radar (SAR) signals on a low-cost, small, lightweight, and low-power consumption platform: Raspberry Pi. The algorithm is implemented with GNU Octave open-source software and the performance was tested on Raspberry Pi 3B and 4 hardware. For performance comparison, a single-threaded baseline implementation of back-projection is created and then modified to run on several threads on an available multicore processor. Executing a single-threaded code Raspberry PI is too slow for real-time imaging. However, the parallelized version shows computation improvement over the baseline version. We include a discussion of parallel implementation on a single Pi using Octave's parallel package. This study contributes to the understanding of implementing SAR image processing on affordable single-board platforms with constrained computing resources.

Keywords: *Synthetic-Aperture Radar (SAR), Image Processing, low-cost, small, lightweight, low-power consumption, Raspberry Pi, Back-projection Algorithm.*

1. Introduction

Synthetic Aperture Radar (SAR) technology has gained increasing significance in various fields, including environmental monitoring, disaster management, agriculture, and defense. It offers the capability to capture high-resolution images of Earth's surface under different weather conditions, making it invaluable for remote sensing applications. However, SAR data processing remains computationally intensive, often requiring specialized hardware or high-performance computing clusters. This poses a challenge, particularly in resource-constrained settings and for researchers seeking cost-effective solutions (Chan, Koo, Chung, & Chuah, 2008; Chang, 2022; Hun et al., 2011; Kurmi, Schedl, & Bimber, 2019; W. K. Lee & Lee, 2017; Li et al., 2017).

The primary objective of this research is to explore the feasibility of implementing the Back-projection SAR image processing algorithm on readily available off-the-shelf low-cost, small, lightweight, low-power consumption platforms. Specifically, we focus on the Raspberry Pi 3 and Raspberry Pi 4, aiming to generate Single Look Complex (SLC) images from the SAR sensor. Our research endeavors to evaluate the performance and limitations of these platforms for SAR data processing and seeks to identify avenues for optimization (Almusawi, Al-Jabali, Khaled, Péter, & Géza, 2022; Saeed, Waqas, Mirbahar, & Khuhro, 2022; Tivnan, Gurjar, Wolf, & Vishwanath, 2015).

This study aims to contribute to the field of signal processing by demonstrating the adaptability and potential of affordable, widely accessible single-board platforms for SAR data processing. By evaluating the performance of the back-projection algorithm on Raspberry Pi 3 and 4, we aim to provide insights into the challenges and opportunities

associated with implementing SAR signal processing on cost-effective hardware. Furthermore, our findings will offer researchers and practitioners valuable information on the capabilities and potential enhancements of such platforms in the domain of SAR image processing.

The foundational principles of SAR signal processing have been extensively explored in the works of (Cumming & Wong, 2005; Richards, 2009). These texts offer a comprehensive understanding of SAR data acquisition, range, and azimuth processing, as well as image formation. The Back-projection algorithm used in this work is widely used due to its advantages in motion error compensation and image reconstruction. Time-domain back projection algorithms, such as global back-projection (GBP), fast back-projection (FBP), and fast factorized back-projection (FFBP), are commonly employed in SAR processing (Ivanenko, Vu, Batra, Kaiser, & Pettersson, 2022; Vu, Sjogren, & Pettersson, 2013).

The utilization of single board computers in signal processing has witnessed a growing body of literature. It is evident that affordable platforms such as Raspberry Pi have found applications in diverse domains. These platforms provide an accessible and cost-effective alternative to traditional high-performance computing clusters (Cardillo, Scandurra, Giusi, & Ciofi, 2021; Carducci, Lipari, Giaquinto, Ponci, & Monti, 2020; Pasolini, Bazzi, & Zabini, 2017; Turicu, Creț, Echim, & Munteanu, 2022).

While SAR processing traditionally demands specialized hardware, researchers like Zhang et al. (2017) have started to explore resource-constrained platforms for this purpose. This aligns with the objectives of this research, which seeks to assess the suitability of Raspberry Pi 3 and 4 for SAR signal processing (Choi, Jeong, Lee, Lee, & Jung, 2021; Cholewa, Pfitzner, Fahnemann, Pirsch, & Blume, 2014; Y. C. Lee, Koo, & Chan, 2017; Schleuniger, Kusk, Dall, & Karlsson, 2013).

The overall goal of this work is to implement the algorithm that maintains the same quality imagery on a platform with low memory and low computing power. This work is presented in three parts: algorithm implementation, algorithm revision, and their practical performance. Section 2 provide details on the hardware and software descriptions, data acquisition method, algorithm implementations, and performance measurements technique. Section 3 presents the results and discussions of the algorithm implementation, followed by a conclusion in section 4.

2. Methodology

This section entailed the utilization of affordable single-board computers, open-source software tools, and the execution of a series of processing steps to achieve the objectives of this research. The hardware, software, and implementation procedures were carefully selected to assess the feasibility of low-cost platforms for SAR image processing.

2.1. Hardware and Software Description

We employed two cost-effective single-board computers, namely the Raspberry Pi 3(RP3) and Raspberry Pi 4(RP4), as the hardware platforms for our SAR signal processing experiments. The Raspberry Pi 3 is equipped with a 1.2 GHz quad-core ARM Cortex-A53 CPU and 1 GB of RAM, while the Raspberry Pi 4 offers improved processing power with a 1.5 GHz quad-core ARM Cortex-A72 CPU and 2 GB of RAM (or 4 GB in some configurations). These compact and affordable devices were chosen due to their wide availability and suitability for embedded computing tasks.

The choice of Raspberry Pi as the testing platform for our research was made after careful consideration of several factors, including performance, hardware availability, and the advantages and limitations of each model—Raspberry Pi 3 and Raspberry Pi 4. We selected the Raspberry Pi 3 for its affordability and established presence in the single-board computer market. It is equipped with a 1.2 GHz quad-core ARM Cortex-A53 CPU and 1 GB of RAM, making it a viable choice for low to moderate processing tasks. While its computational capabilities are relatively modest, it offers a balanced performance that aligns with our objective of evaluating SAR signal processing on cost-effective hardware.

The Raspberry Pi 4 was chosen for its superior performance, which was a significant advantage in our research. With a 1.5 GHz quad-core ARM Cortex-A72 CPU and 8 GB of RAM, it offers substantial processing power for a single-board computer. This model's improved specifications were especially valuable when dealing with SAR data, which can be memory and computationally demanding.

Both Raspberry Pi 3 and Raspberry Pi 4 are widely available and accessible to researchers, educators, and hobbyists, which was a key consideration in our choice. Their popularity has resulted in a robust ecosystem of software and support, making them suitable platforms for experimentation and development. Raspberry Pi 3 has high affordability, energy efficiency, and compatibility with a range of accessories and peripherals. However, it has modest processing power, which could be a constraint when handling large SAR datasets or demanding signal processing operations.

Raspberry Pi 4 has enhanced performance, higher RAM capacity, and better support for more compute-intensive tasks. Its capabilities were especially beneficial for our SAR signal processing experiments. While it outperforms the Raspberry Pi 3, it is still a single-board computer with inherent limitations compared to more powerful computing platforms. Additionally, it generates more heat and consumes more power than its predecessor.

The choice of Raspberry Pi 3 and Raspberry Pi 4 as our testing platforms was driven by a balance between affordability, performance, and hardware availability. The Raspberry Pi 3 provided a cost-effective option for assessing SAR signal processing, while the Raspberry Pi 4's improved hardware specifications made it an attractive choice for tasks demanding more computational power. This selection allowed us to comprehensively evaluate the potential of these affordable platforms in the context of SAR signal processing, considering their respective advantages and limitations. Table 1 shows the specifications of the hardware that we use in this work.

Table 2-1: Hardware Specification

Specifications	**Raspberry Pi 3B+**	**Raspberry Pi 4**
CPU	Quad-core ARM Cortex-A53 @ 1.4 GHz	Quad-core ARM Cortex-A72 @ 1.5 GHz
RAM	1 GB	4 GB
USB	4 x USB 2.0	2 x USB 3.0
Network	Gigabit Ethernet	Gigabit Ethernet
HDMI Port	1 x HDMI 1.4	2 x Micro HDMI (HDMI 2.0)
HDMI Resolution	1080p	4K @ 60Hz or 1080p @ 60Hz
GPIO	Yes	Yes
Graphics	40-pin	40-pin

The software stack used in this research predominantly relies on open-source solutions. We harnessed the power of the GNU Octave software as our primary development tool for the signal and image processing code. Additionally, the Raspberry Pi platforms operated on the Raspbian operating system, a Debian-based Linux distribution optimized for these single-board computers. The open-source nature of these software components aligns with our objective of affordability and accessibility in SAR signal processing.

2.1. Data Acquisition

The raw signal data that we used in this work was the results of a SAR measurement campaign conducted inside a hangar with a size of 25 by 25 meters. The experimental SAR radar consisted of a PC connected to a portable VNA with two horn antennas (Figure 2-1). The radar was mounted at a height of 5 meters (Figure 2-2) with the horn antenna facing several objects as the targets, such as corner reflectors and small UAVs (Figure 3-1).

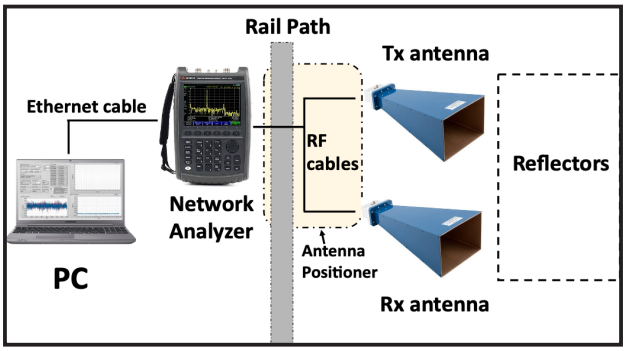


Figure 2-1: Basic experimental SAR hardware configuration.

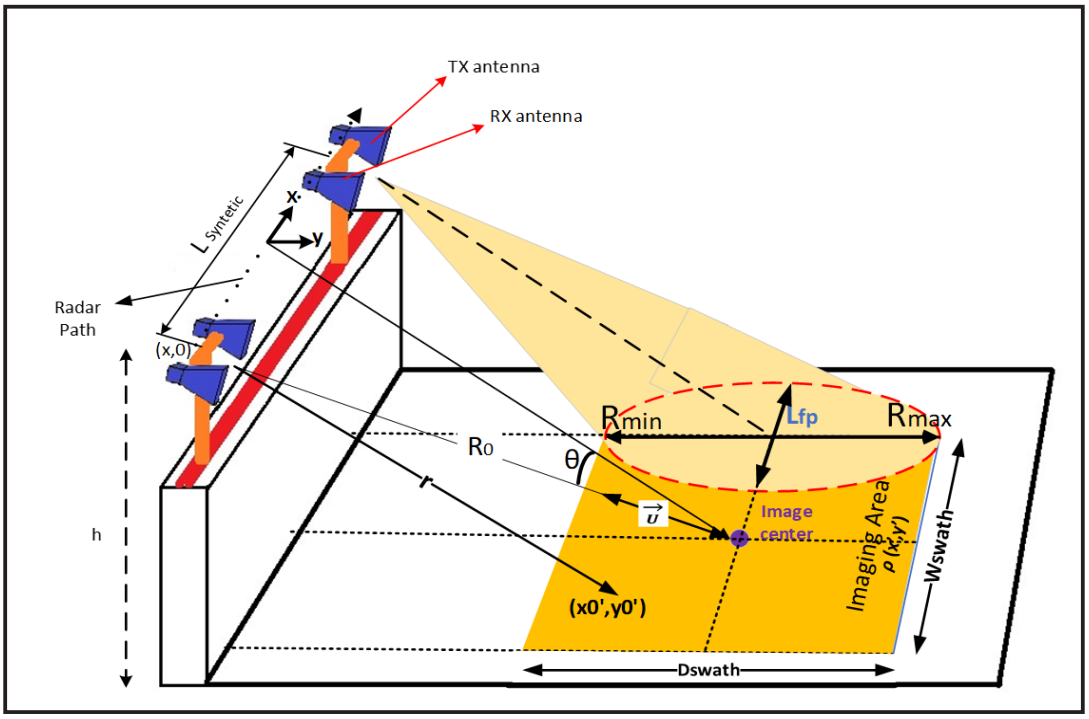


Figure 2-2: SAR data acquisition geometry

The configuration of the basic ground-based SAR scanning system is shown in Figure 2-2. The reflection function of the targets in the scanning region of a two-dimensional (2D) imaging geometry is defined as $\rho(x, y)$, and the antenna positions are expressed in Cartesian coordinates as $(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{D_p \sin\theta}, \boldsymbol{D_p \cos\theta})$ where, $\boldsymbol{D_p}$ and $\boldsymbol{\theta}$ are the distance and angle between the antennas and the scene origin, respectively. The radar's backscatter signal can be described as follows with $\boldsymbol{PP}$ as the total number of scatterers.

$$\boldsymbol{S_\beta(k) = \sum_{1}^{P} \rho_p e^{-jkD_p(\theta)}} \qquad \textbf{(2-1)}$$

where, $k = 4\pi f / c$ is a (two-way) wave number, f and c are the radar frequency and speed of the light, respectively.

The instantaneous distance between the targets and antennas is given by

$$\boldsymbol{D_p(\beta) = \sqrt{(x - x_p)^2 + (y - y_p)^2}} \qquad \textbf{(2-2)}$$

where the coordinates of the p-th scatterer and antennas are represented by (x_p, y_p) and (x, y) respectively.

As demonstrated in Figure 2-2 the SAR platform gathers 2D GBSAR data by gathering back-scatter data at a total of N distinct places along the L synthetic aperture. Next, focused complex SAR data $\rho(x, y)$ is obtained using any reconstruction algorithm, such as back-propagation algorithm (BPA), to convert the received raw SAR data into a comprehensible image (Yigit, 2013; Yigit, Demirci, & Ozdemir, 2021).

Table 2-2 shows the parameters used in the measurement using VNA.

Table 2-2: Data Acquisition Parameters

Specifications	**Value**
Frequency	5 to 7 GHz
S Parameters	S21
Pulse Type	SFCW
Frequency steps	0.5 MHz
Number of frequency samples	4001
Height	5 m
Number of cross range samples	173
Nominal transmission power	-1 dBm
Polarization	HV
Cross range sample sampling	10 cm

2.2. Back-projection Algorithm Theory for Image Formation

In this study, the SAR signal processing system is realized by applying the back-projection algorithms as outlined in (Yigit, 2013). The development of the Back-projection Algorithm (BPA) commences with the examination of the inverse Fourier transform (IFT) expression of the reflectivity function $\boldsymbol{\rho(x', y')}$, which is provided in Cartesian coordinates.

$$\rho(x', y') = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} G(k_x, k_y) \cdot e^{j(k_x x + k_y y)} \mathrm{d}k_x \mathrm{d}k_y \qquad \textbf{(2-3)}$$

In the provided equation, $G(k_x, k_y)$ corresponds to the 2-D Fourier transform (FT) of $\rho(x', y')$. As illustrated in Figure 2-2, the instantaneous location of the antenna platform at (x, y) is determined by a unit vector $\vec{u}$ that originates from the center of the scene and points towards this specific position. The associated observation angle at this location is denoted as θ, representing the angle between the unit vector $\vec{u}$ and the range axis y. Since the spatial frequency data samples are collected along the polar radius θ, as depicted in Figure 2-2, Equation (2-3) can be adjusted and expressed in polar coordinates (k_r, θ) as follows:

$$\rho(x', y') = \int_{-\pi}^{+\pi} \int_{0}^{+\infty} G(k_r, \theta) \cdot \exp(j k_r r) k_r \mathrm{d}k_r \mathrm{d}\theta \qquad \textbf{(2-4)}$$

In this context, r is the range from an instantaneous antenna platform location to the point$(x_0', y_0')(x_0', y_0')$. The projection-slice theorem is employed to establish a connection between the target's FT $G(k_x, k_y)$ and the available measured data $S_\theta(k_r)$. In a 2-D space, the theorem essentially states that one-dimensional (1-D) Fourier Transform of the projection at the angle θ represents the slice of the 2-D FT of the projected (original) scene at the same angle, i.e., $S_\theta(k_r) \equiv G(k_r, \theta)$. Consequently, the sampled representation of $G(k_x, k_y)$ can be derived from the Fourier Transform of the projections $S_\theta(k_r)$ measured at various observation angles. Using this principle, Eq. (2-4) can be expressed as below:

$$\rho(x', y') = \int_{-\pi}^{+\pi} \left[\int_{0}^{+\infty} S_\theta(k_r) \exp(j k_r r) k_r \mathrm{d}k_r \right] \mathrm{d}\theta \qquad \textbf{(2-5)}$$

The integral enclosed in Eq. (2-5) can be considered as the 1-D Inverse Fourier Transform (IFT) of a function $q_\theta(r) = S_\theta(k_r)\, k_r\, q_\theta(r) = S_\theta(k_r)\, k_r$ evaluated at the distance r. By defining

$q_\theta(r)\, q_\theta(r)$ as the IFT of this function, Eq. (2-5) can be represented as

$$\rho(x', y') = \int_{-\pi}^{\pi} q_\theta(r) \mathrm{d}\theta. \quad \textbf{(2-6)}$$

Equation (2-6) represents the ultimate focused image obtained from the 2-D-filtered Back-projection algorithm. projection algorithm. projection algorithm.

3. Result and Discussion

To conduct SAR imaging experiments, the measurement system given in Figure 2-1 was constructed in the Research Center for Aviation Technology, National Research and Innovation Agency, Rumpin, Bogor. The system consists of two C Band horn antennas (WR-159), a Vector network analyzer (N9917A) and computer to collect the phase histories data. Measurements were carried out in the frequency range of 5 - 7 GHz. Horizontal half-power beam width of the antennas was 16.9 degree and vertical at 14.3 degree with 20 dB Gain. The picture of the objects used as target are given in Figure 2-2 with the exact location of each object in Figure 3-2.

3.1. Ground Based SAR Experiment

The SAR scenario for obtaining backscattering data from targets. The targets were set on the ground parallel to the scanning path. P1 through P6 were the spatial coordinates of the six reflectors. Four corner reflectors of varied diameters were used to establish a reference for the radar cross-section (RCS).

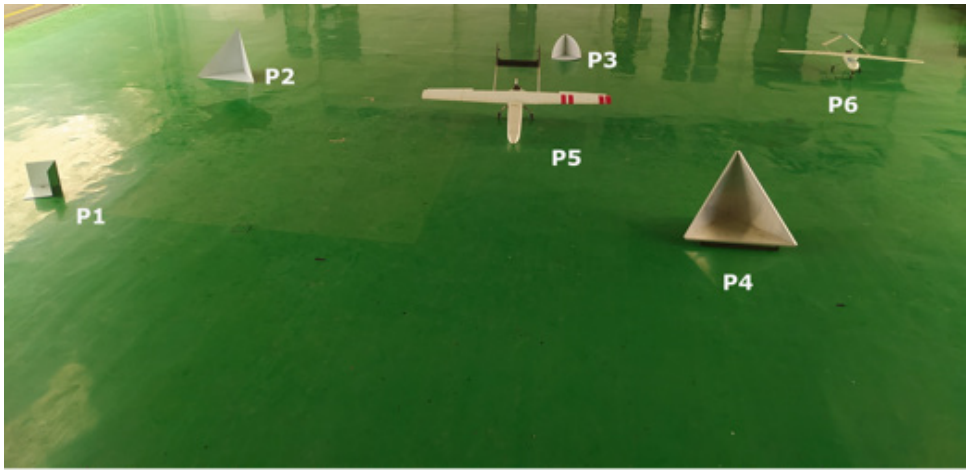


Figure 3-1: Photograph of the radar targets scene

The dimensions of these reflectors are as follows: P1 has a side length of 40 cm, P2 and P4 have a side length of 100 cm, and P3 has a side length of 50 cm. Furthermore, P5 and P6, a prototype of an Unmanned Aerial Vehicle (UAV) featuring a fuselage constructed from composite material, also contribute to the reflection process.

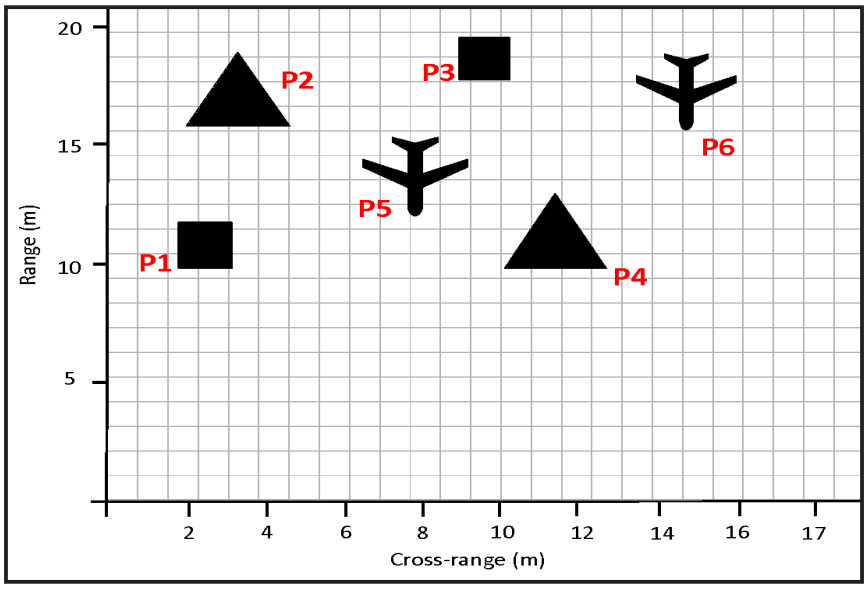
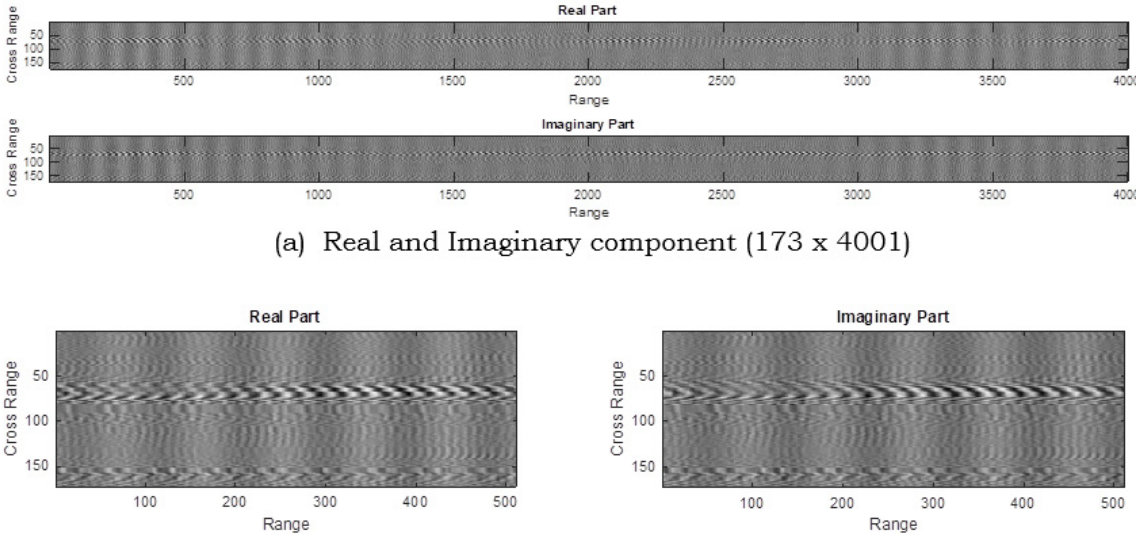


Figure 3-2: Target Locations

Figure 3-3 shows the real and imaginary part of the raw data acquired from the measurement data of S21 by the VNA. The measurement was taken once at each azimuth position, resulting in 173 azimuth bins consisting of 4001 range samples. The complex IQ data then organized into a complex floating-point matrix with a size of 173x4001 samples.



(a) Real and Imaginary component (173 x 4001)

Figure 3-3: The real and imaginary component of the SAR complex signal measured from the experiment.

To confirm the data acquisition correctness, we preprocessed the phase histories data using inverse Fourier transform to produce the range profile in Figure 3-4. In the picture, it can be observed that there are backscatter values in the form of arcs at sample ranges 167, 162, 205, 235, 264, and 272.

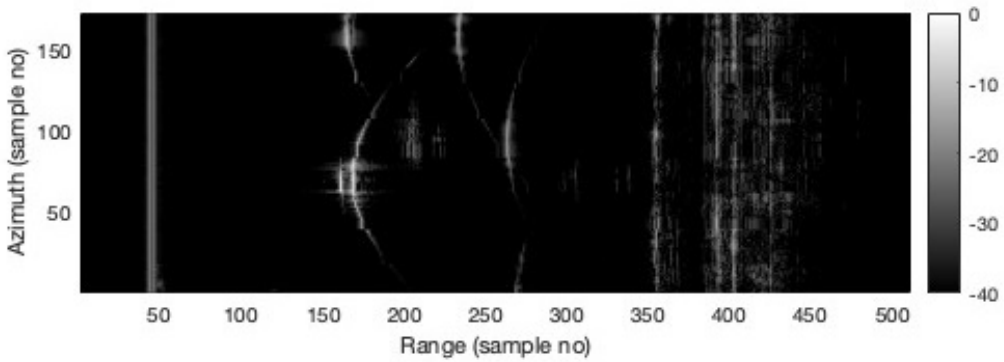


Figure 3-4: Magnitude image of range compressed data (dB).

3.2 Back-projection Algorithm Implementation

The back-projection algorithm from section 2.3 is implemented as the following:

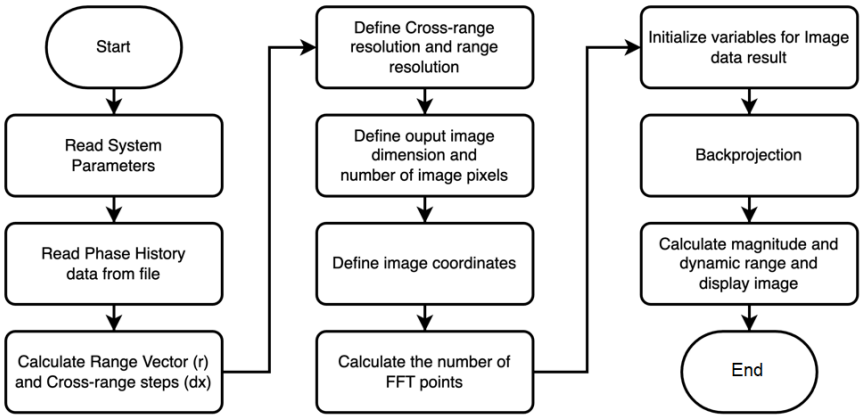


Figure 3-5: Flow chart for main program block of back-projection.

First, the program reads system parameters such as wave propagation speed, system geometry, pulse number, and resulting image size. The program then begins by loading phase history data from an external file obtained from SAR data acquisition. The loaded data is stored in a variable which will be processed further in the subsequent steps.

The processing continues with adjustments to the phase history data. The orientation of the frequency vector is ensured to be horizontal. Range vectors are calculated, and parameters like the maximum range and cross-range sample spacing are determined. The program then defines the cross-range resolution and range resolution, crucial for the later steps of the imaging process. Image dimensions are defined, along with the number of image pixels. The coordinates of the final image are established. The number of Fast Fourier Transform (FFT) points is computed for later data transformations.

The core of the back-projection program involves looping through each pulse in the data as shown in Figure 3-6. For each pulse, an Inverse Fast Fourier Transform is applied to the complex data. The required distances and angles for each image point are calculated. Points within a specified angle range are identified, and the image data is updated accordingly.

After processing all pulses, the program creates the final SAR image. The data is transformed and scaled using logarithmic operations, ensuring the dynamic range of the image is manageable and visually informative. The resulting image is ready for display and further analysis.

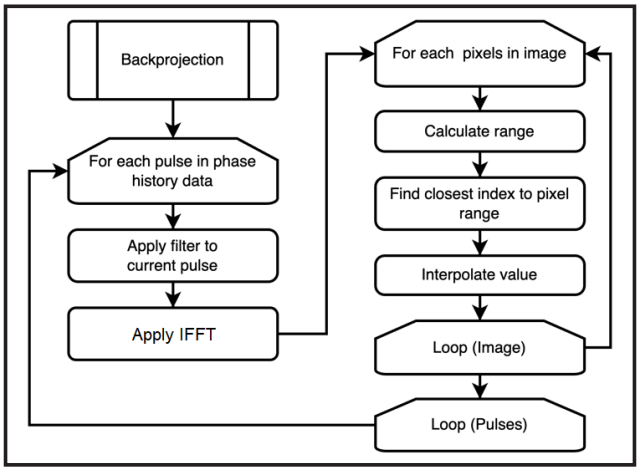


Figure 3-6: Flow chart for main program block of back-projection.

The implementation this flowchart to the Back-projection algorithm mentioned in equation theory section 2. can be described as follows:

1. For all observation angles repeat step 2 to 5.
2. Take 1-D IFT of $\boldsymbol{S_\beta(k_r)}$. to obtain $\boldsymbol{q_\beta(r)}$.
3. For each pixel, calculate the corresponding range value r.
4. For each pixel, calculate $\boldsymbol{q_\beta(r)}$ value through interpolation.
5. Add interpolated values to $\boldsymbol{\rho(x', y')}$.

3.3 Execution Time

To demonstrate and analyze the performance of the back-projection program, we have implemented two distinct versions using the GNU Octave environment. The first version represents the baseline implementation, which exclusively employs a single thread operating on a single processor core. In contrast, the second version utilizes multiple threads, thus enabling parallel execution across multiple processor cores.

These implementations facilitate a comprehensive evaluation of the program's efficiency and scalability in leveraging modern multi-core processor architectures used in the hardware. The baseline version serves as a reference point for assessing the benefits of parallelization in the second version, shedding light on the potential performance gains achieved through multi-threaded processing.

3.3.1 Single Thread Version

To evaluate the baseline version of the program, input data of dimensions 173x4001 (cross-range x range) samples will be processed to generate four distinct images of varying sizes: 128x128, 256x256, 512x512, and 1024x1024. Each sample in both the input and output data uses the complex double data type, which is implemented in GNU Octave and consumes 8 bytes per sample. This testing strategy allows for a comprehensive assessment of the program's performance across different output image sizes, reflecting real-world scenarios where users might require various resolutions for specific applications. By employing complex double data types and accurately accounting for the memory consumption of 8 bytes per sample, the testing framework ensures an accurate representation of the program's resource utilization and computational demands, which are vital considerations in scientific and computational research.

Table 3-1 and Figure 3-6 present the program execution times, which were measured by introducing code lines to record the time before and after the back-projection process. The table highlights the execution times for baseline version when processing input data to generate images of various dimensions.

Table 3-1: Execution Time (seconds) of BPA on Raspberry Pi (RPI) 3 and 4 for Image Size of 128x128, 256x256, 512x512, 1024x1024

Image Size	RPI 3 (s)	RPI 4 (s)
128 x 128	6.6652	3.8099
256 x 256	13.6745	9.5814
512 x 512	42.7172	29.7940
1024 x 1024	156.9362	116.6527

The algorithm allows for the image to be constructed step by step, focusing on one pulse at a time. This means that only the current pulse being processed and the image itself needs to be stored in memory at any given moment. The benefit of reduced memory usage comes with a tradeoff – there's a noticeable increase in the computational workload.

To break it down, the computations required for filtered back-projection can be summarized as follows: For a data collection with P azimuth samples (pulses) and L frequency bins (samples) for each pulse, the convolution step (Fourier multiplication) involves $\boldsymbol{O(PL\,log(L))}$ operations. The data is then used in the back-projection step, essentially adding up the results of the convolution step for each pixel in the image.

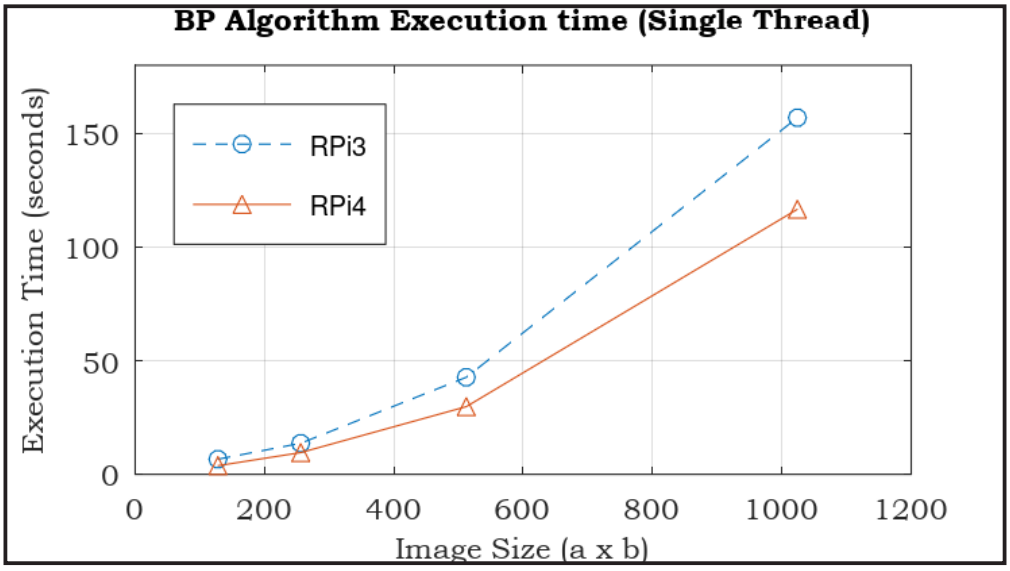


Figure 3-7: Single Thread BPA execution time for image size: 128x128, 256x256, 512x512 and 1024x1024

Based on single thread execution time data from Table 3-1, if we compare the execution time between the same image size for RPI4 and RPI3, we can see that for image size 128x128, 256x256, 512x512 and 1024x1024, RPI 4 has 1.74, 11.42, 1.43 and 1.34 times faster than its counterpart. These measurements are also in line with the facts that for an image with N × N pixels and P projections, the back-projection step requires $\boldsymbol{O(PN^2)}$ operations. Combining these two steps results in a total number of operations $\boldsymbol{O(PL\,log(L) + PN^2)}$. If N is significantly larger than L, N^2 dominates the term, and the required operations can be approximated as $\boldsymbol{O(PN^2)}$. In many cases, it's assumed that P = N, making the computational load around $\boldsymbol{O(N^3)}$. This represents a substantial increase in the number of operations (for large P) (Rasmussen 2016).

For a verification of the algorithm result four result images are shown in Figure 3-7.

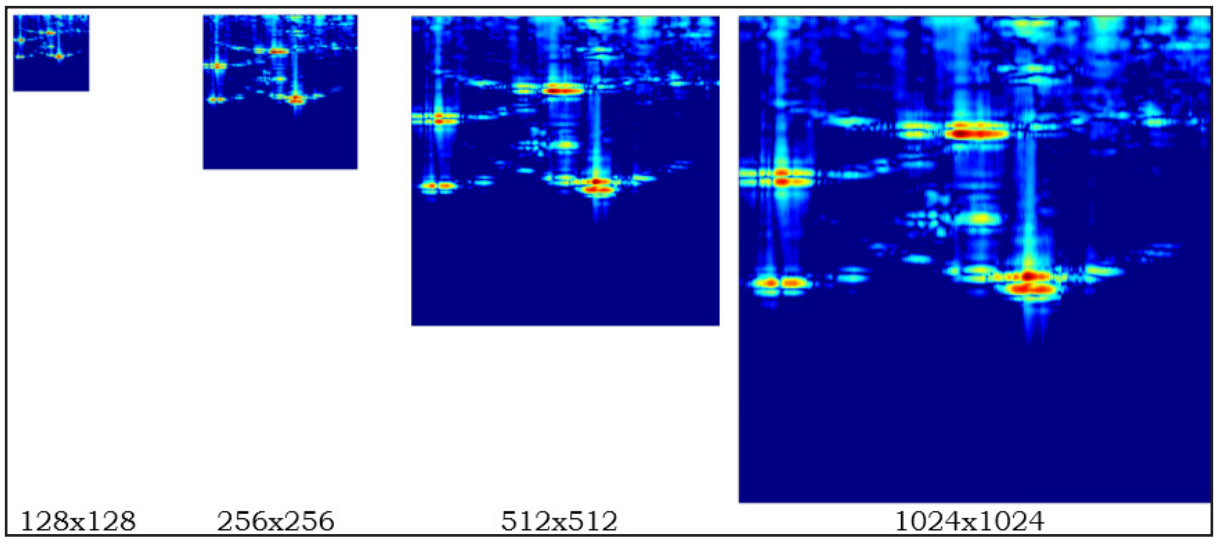


Figure 3-7: Single Thread BP algorithm result for images size: 128x128, 256x256, 512x512 and 1024x1024

Figure 3-7 shows the results of focusing using the BPA for various image sizes. These results demonstrate the backscatter values consistent with the objects visible in Figures 3-1 and 3-2. They also indicate that the algorithm's implementation on the GNU Octave platform is functioning correctly.

3.3.2 Multithread Version

To harness the available multicore processors on the Raspberry Pi, the baseline program, initially designed to use a single thread, was adapted to a multithreaded approach. We leveraged a library package called "Parallel," which is available in GNU Octave, to create a multithreaded version of this algorithm's implementation. The "Parallel" package offers the *pararrayfun* function, which serves to execute the same program routine on diverse datasets, following the principle of "Single Instruction Multiple Data" (SIMD).

This modification allowed us to take full advantage of the multicore architecture of the Raspberry Pi, enhancing the program's performance by parallelizing the processing of different data subsets. The "Parallel" package's *pararrayfun* function efficiently orchestrates the execution of the algorithm across multiple threads, further optimizing computational resources and execution times.

Flow chart of the multithreaded version of BPA is shown in Figure 3-8.

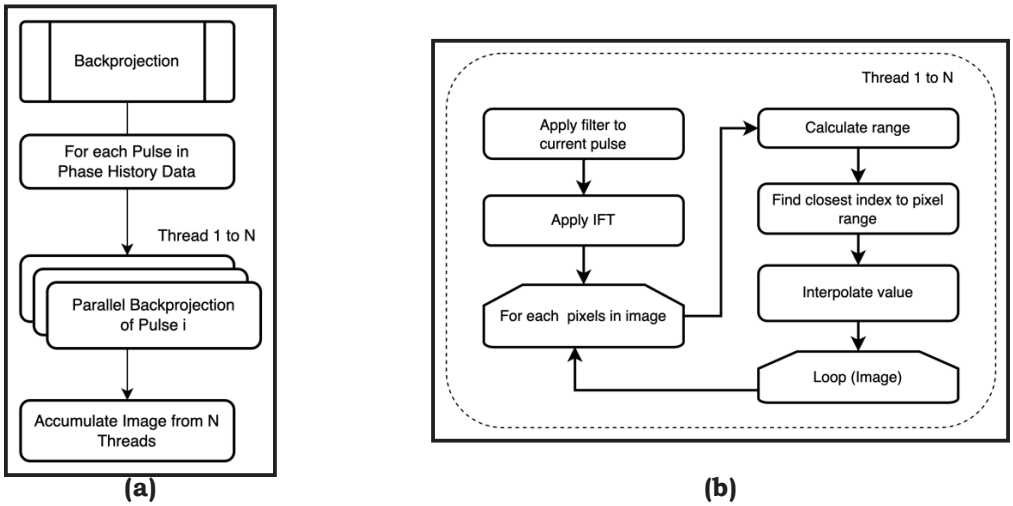


Figure 3-8: Flow chart for Parallel Back-projection implementation (a) Flow chart for general processing of the algorithm and (b) Flow chart for the parallel processing of each pulse that run in Thread 1 to N.

The parallelization strategy involves transforming the sequential processing of pulses into a parallel approach, matching the number of processor cores available. In this manner, since the Raspberry Pi has 4 processor cores, the maximum number of threads that will be executed in parallel is 4. To demonstrate the impact of the number of threads on data processing speed, we recorded the program's execution speed for four different dataset sizes, each executed using one to four threads. In this experiment, we aimed to illustrate how varying the number of threads influences the processing speed of data. By testing different dataset sizes across a range of threads, we could observe the correlation between the number of threads and the program's overall execution time, providing valuable insights into the program's performance under different configurations. Table 3-2 shows the execution time of the multithreaded BPA.

Table 3-2: Execution time (seconds) of the BPA on two platforms considering image size (128x128, 256x256, 512x512, 1024x1024) and number of threads (1,2,3,4)

		RPI3 (s)				RPI4 (s)			
		Image Size							
		128x128	256x256	512x512	1024x1024	128x128	256x256	512x512	1024x1024
No of Threads	1	6.6652	13.6745	42.7172	156.9362	3.8099	9.5814	29.7940	116.6527
	2	4.1642	9.3552	20.8944	84.3915	2.5791	5.7540	17.2301	66.0177
	3	3.4395	8.4475	17.5016	65.4754	2.3223	4.7237	13.4965	50.5040
	4	2.3426	6.4746	15.8629	54.2575	2.9596	4.4214	12.0937	44.1094

From Figure 3-9 can be seen that for each image size, as the number of threads is increasing the execution time is decreasing (processing getting faster).

To calculate the speed-up due to the addition of threads, we can compare the execution time of the program for each number of threads with the execution time of the program running with a single thread for each image size. Table 3-3 displays the speed-up values for both platforms, various image sizes, and the number of threads used. In general, the larger the number of threads used, the larger the speed-up value become. In terms of image size with the same number of threads, the speed-up value increases as the image size grows.

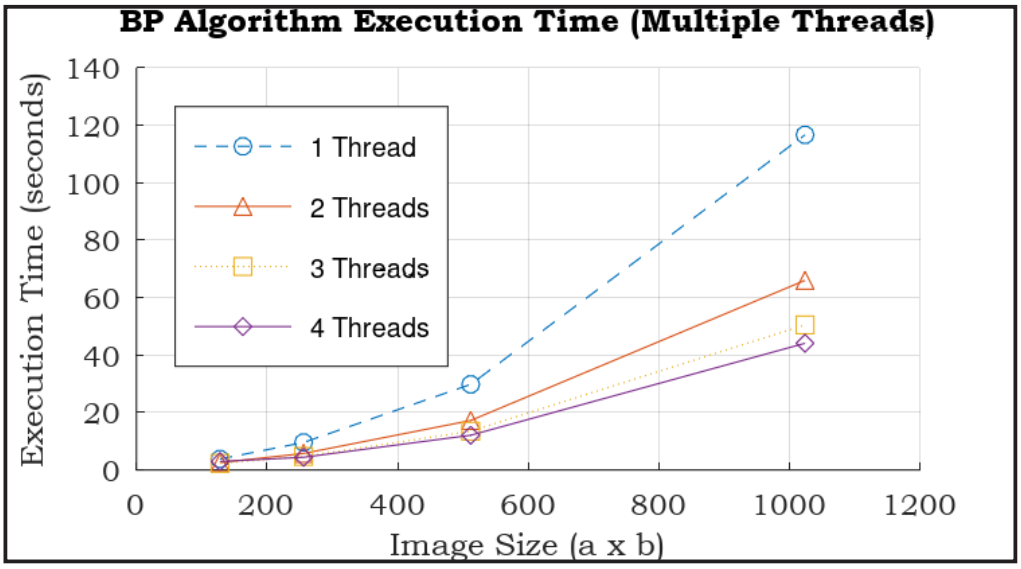


Figure 3-9: Multiple Thread BPA execution time using RPI4 for image size: 128x128, 256x256, 512x512, and 1024x1024 and thread size 1, 2, 3, and 4.

Comparing the performance of RPI3 and RPI4, although in terms of execution time RPI4 is faster than RPI3, however speed-up values of RPI4 for each number of threads are lower than its counterpart.

Table 3-3: Execution time (seconds) of the BPA on two platforms considering image sizes (128x128, 256x256, 512x512, 1024x1024) and number of threads (1,2,3,4)

		RPI3 (s)				RPI4 (s)			
		Image Size							
		128x128	256x256	512x512	1024x1024	128x128	256x256	512x512	1024x1024
No of Threads	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.60	1.46	2.04	1.86	1.48	1.67	1.73	1.77
	3	1.94	1.62	2.44	2.40	1.64	2.03	2.21	2.31
	4	2.85	2.11	2.69	2.89	1.29	2.17	2.46	2.64

4. Conclusions

We can conclude that the Back-Projection Algorithm for filtered back-projection on Raspberry Pi 4 (RPI4) is faster than its counterpart Raspberry Pi 3 (RPI3) for all image sizes and number of threads. Furthermore, our research on the implementation of the BPA for SAR signal processing on cost-effective single board computers, specifically the Raspberry Pi 3 and Raspberry Pi 4, has shed light on the following key conclusions: The Raspberry Pi 4 exhibited superior computational performance compared to the Raspberry Pi 3, making it a more attractive choice for SAR signal processing tasks, particularly with smaller datasets. The limitations identified, especially with larger datasets, emphasize the need for optimization techniques like parallelization to enhance the processing speed on these platforms, providing an exciting avenue for future research.Overall, this study contributes to the understanding of leveraging affordable, off-the-shelf hardware for SAR signal processing, while underscoring the importance of optimization in realizing the full potential of these platforms. These insights offer researchers, educators, and practitioners valuable guidance for achieving efficient SAR applications with limited resources, ultimately advancing the accessibility and affordability of SAR technology.

Acknowledgements

This study is supported by the Research Centre for Aeronautics Technology, National Research and Innovation Agency (BRIN) Indonesia, as part of the Research Program: Synthetic Aperture Radar (SAR) on an Unmanned Aerial Platform. We are also funded by the Lembaga Pengelola Dana Pendidikan (LPDP) as part of the Riset dan Inovasi untuk Indonesia Maju (RIIM) program. Furthermore, we thank the Department of Electrical Engineering at Universitas Indonesia for the facilities and technical assistance provided.

Contributorship Statement

All authors contributed equally.

References

Almusawi, H. A., Al-Jabali, M., Khaled, A. M., Péter, K., & Géza, H. (2022). Self-Driving robotic car utilizing image processing and machine learning. *IOP Conference Series: Materials Science and Engineering, 1256*(1), 012024. doi:10.1088/1757-899X/1256/1/012024

Benson, T. M., Campbell, D. P., & Cook, D. A. (2012, 7-11 May 2012). *Gigapixel spotlight synthetic aperture radar back-projection using clusters of GPUs and CUDA*. Paper presented at the 2012 IEEE Radar Conference.

Capozzoli, A., Curcio, C., & Liseno, A. (2013). Fast Gpu-Based Interpolation for SAR Back-projection. *Progress in Electromagnetics Research-pier, 133*, 259-283.

Cardillo, E., Scandurra, G., Giusi, G., & Ciofi, C. (2021). A Two-Channel DFT Spectrum Analyzer for Fluctuation Enhanced Sensing Based on a PC Audio Board. *Sensors, 21*(13), 4307. Retrieved from https://www.mdpi.com/1424-8220/21/13/4307

Carducci, C. G. C., Lipari, G., Giaquinto, N., Ponci, F., & Monti, A. (2020). Error Model in Single-Board Computer-Based Phasor Measurement Units. *IEEE Transactions on Instrumentation and Measurement, 69*(9), 6155-6164. doi:10.1109/TIM.2020.2967245

Chan, Y. K., Koo, V., Chung, B., & Chuah, H.-T. (2008). Modified algorithm for real time sar signal processing. *Progress in Electromagnetics Research C, 1*, 159-168. doi:10.2528/PIERC08021801

Chang, R. (2022). Development and application of data mining method for synthetic aperture radar image ship inspection based on big data application technology. *Journal of Physics: Conference Series, 2294*(1), 012006. doi:10.1088/1742-6596/2294/1/012006

Choi, Y., Jeong, D., Lee, M., Lee, W., & Jung, Y. (2021). FPGA Implementation of the Range-Doppler Algorithm for Real-Time Synthetic Aperture Radar Imaging. *Electronics, 10*(17), 2133. Retrieved from https://www.mdpi.com/2079-9292/10/17/2133

Cholewa, F., Pfitzner, M., Fahnemann, C., Pirsch, P., & Blume, H. (2014, 13-17 Oct. 2014). *Synthetic aperture radar with back-projection: A scalable, platform independent architecture for exhaustive FPGA resource utilization.* Paper presented at the 2014 International Radar Conference.

Cruz, H., Véstias, M., Monteiro, J., Neto, H., & Duarte, R. P. (2022). A Review of Synthetic-Aperture Radar Image Formation Algorithms and Implementations: A Computational Perspective. *Remote Sensing, 14*(5). doi:10.3390/rs14051258

Cumming, I. G., & Wong, F. H. (2005). *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation.*

Fasih, A., & Hartley, T. (2010, 10-14 May 2010). *GPU-accelerated synthetic aperture radar back-projection in CUDA.* Paper presented at the 2010 IEEE Radar Conference.

Gorham, L. A., & Moore, L. J. (2010). SAR image formation toolbox for MATLAB. *SPIE Proceedings, 7699*, 769906. doi:10.1117/12.855375

Hu, R., Mysore, B. S., Alaee-Kerahroodi, M., & Ottersten, B. (2020). Orthorectified Polar Format Algorithm for Generalized Spotlight SAR Imaging with DEM. *IEEE Transactions on Geoscience and Remote Sensing, 59*, 1-9. doi:10.1109/TGRS.2020.3011638

Hun, L., Lim, C., Chua, M. Y., Chan, Y. K., Lim, T. S., & Koo, V. (2011). A new data acquisition and processing system for UAVSAR. *IEICE Electronics Express, 8*, 1716-1722. doi:10.1587/elex.8.1716

Ivanenko, Y., Vu, V. T., Batra, A., Kaiser, T., & Pettersson, M. I. (2022). Interpolation Methods with Phase Control for Back-projection of Complex-Valued SAR Data. *Sensors, 22*(13), 4941. Retrieved from https://www.mdpi.com/1424-8220/22/13/4941

Kurmi, I., Schedl, D. C., & Bimber, O. (2019). A Statistical View on Synthetic Aperture Imaging for Occlusion Removal. *IEEE Sensors Journal, 19*(20), 9374-9383. doi:10.1109/JSEN.2019.2922731

Lee, W. K., & Lee, K. W. (2017). Experimental operation of drone micro-SAR with efficient time-varying velocity compensation. *Electronics Letters, 53*(10), 682-683. doi:https://doi.org/10.1049/el.2017.0801

Lee, Y. C., Koo, V. C., & Chan, Y. K. (2017, 19-22 Nov. 2017). *Design and development of FPGA-based FFT Co-processor for Synthetic Aperture Radar (SAR).* Paper presented at the 2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL).

Li, Z., Su, D., Zhu, H., Li, W., Zhang, F., & Li, R. (2017). A Fast Synthetic Aperture Radar Raw Data Simulation Using Cloud Computing. *Sensors, 17*(1), 113. Retrieved from https://

www.mdpi.com/1424-8220/17/1/113

Pasolini, G., Bazzi, A., & Zabini, F. (2017). A Raspberry Pi-Based Platform for Signal Processing Education [SP Education]. *IEEE Signal Processing Magazine, 34*(4), 151-158. doi:10.1109/MSP.2017.2693500

Richards, J. A. (2009). *Remote Sensing with Imaging Radar*: Springer Publishing Company, Incorporated.

Rasmussen, A. (2016). Implementation and Performance of Factorized Back-projection on Low-cost Commercial-Off-The-Shelf Hardware. Air Force Institute of Technology Theses and Dissertations.

Saeed, U., Waqas, M., Mirbahar, N., & Khuhro, M. (2022). Comparative analysis of different Operating systems for Raspberry Pi in terms of scheduling, synchronization, and memory management. *Mehran University Research Journal of Engineering and Technology, 41*, 113-119. doi:10.22581/muet1982.2203.11

Schleuniger, P., Kusk, A., Dall, J., & Karlsson, S. (2013). *Synthetic Aperture Radar Data Processing on an FPGA Multi-core System*, Berlin, Heidelberg.

Tivnan, M., Gurjar, R., Wolf, D. E., & Vishwanath, K. (2015). High Frequency Sampling of TTL Pulses on a Raspberry Pi for Diffuse Correlation Spectroscopy Applications. *Sensors, 15*(8), 19709-19722. doi:10.3390/s150819709

Turicu, D. C., Creţ, O., Echim, M., & Munteanu, C. (2022). An FPGA-Based Solution for Computing a Local Stationarity Measure From Satellite Data. *IEEE Access, 10*, 9668-9676. doi:10.1109/ACCESS.2022.3143239

Ulander, L. M. H., Hellsten, H., & Stenstrom, G. (2003). Synthetic-aperture radar processing using fast factorized back-projection. *IEEE Transactions on Aerospace and Electronic Systems, 39*(3), 760-776. doi:10.1109/TAES.2003.1238734

Vu, V. T., Sjogren, T. K., & Pettersson, M. I. (2013). Fast Time-Domain Algorithms for UWB Bistatic SAR Processing. *IEEE Transactions on Aerospace and Electronic Systems, 49*(3), 1982-1994. doi:10.1109/TAES.2013.6558032

Yigit, E. (2013). Short-range ground-based synthetic aperture radar imaging: performance comparison between frequency-wavenumber migration and back-projection algorithms. *Journal of Applied Remote Sensing, 7*, 073483. doi:10.1117/1.JRS.7.073483

Yigit, E., DemİRcİ, Ş., & Ozdemir, C. (2021). Clutter removal in millimeter wave GB-SAR images using OTSU's thresholding method. *International Journal of Engineering and Geosciences, 7*. doi:10.26833/ijeg.867467

$\beta q_\theta(r)$