International Journal of Aeronautical and Space Sciences
https://doi.org/10.1007/s42405-025-00930-w

IJASS International Journal of Aeronautical and Space Sciences

ORIGINAL PAPER

Parallelized Monte Carlo Optimization with Efficient Pipelining on an FPGA for Onboard Ballistic Target Tracking

Seongjin Yoon[1] · Heoncheol Lee[2] · Hyuckhoon Kwon[3] · Bora Jung[3] · Wonseok Choi[3]

Received: 13 September 2024 / Revised: 19 February 2025 / Accepted: 4 March 2025

Abstract
This study addresses the challenge of accurately estimating high-speed ballistic targets in complex environments. To achieve precise target tracking in nonlinear and non-Gaussian noise conditions, Monte Carlo optimization (MCO) is employed. While increasing the number of samples in MCO enhances target estimation accuracy, it simultaneously escalates the computational burden, which in turn significantly extends processing time, making real-time application difficult. To overcome this limitation, we propose a parallelized MCO algorithm implemented on a field-programmable gate array (FPGA). The proposed approach achieves a reduction in computation time by up to 2.95× compared to conventional MCO methods. Furthermore, it delivers a 5.75× decrease in power consumption relative to GPU-based MCO implementations.

Keywords Monte Carlo optimization · Parallelization · FPGA · Target tracking

1 Introduction

In high-speed target tracking, the real-time estimation of the position and motion states of the target is critical. The Kalman Filter (KF) remains the most widely implemented algorithm for target tracking, particularly in linear systems with Gaussian noise characteristics. However, real-world systems often exhibit nonlinear and non-Gaussian behaviors, rendering the KF suboptimal in such scenarios [1, 2]. Consequently, advanced optimization algorithms, including the Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), Particle Filter (PF), and Monte Carlo Optimization (MCO), are better suited for these conditions [3, 4].

This study focuses on the application of MCO, which is particularly effective for state estimation using random sampling. While AI-driven deep learning methods excel in estimation and tracking applications, they require a large amount of training data [5, 6]. In guided missile applications, obtaining sufficient real-world training data for deep learning models is extremely challenging due to military confidentiality. Given these limitations, MCO was selected as it provides a robust and adaptable solution that does not require pre-trained models or large datasets. Instead, it operates using random sampling techniques, making it well-suited for real-time state estimation in unpredictable conditions.

However, MCO introduces specific constraints that must be addressed:

Uniform Distribution of Random Numbers: Accurate state estimation requires a large set of uniformly distributed random numbers. Any deviation from uniformity can introduce bias, adversely affecting the optimization process.

Increased Computational Load: While increasing the number of samples and iterations enhances algorithmic accuracy, it also significantly escalates the computational burden. This increase in computational demand poses challenges for real-time target tracking, as it can lead to critical delays.

In our previous work [7], we addressed the computational demands of MCO by leveraging Graphics Processing Units (GPUs) to achieve significant speed improvements. However, missile guidance systems operate in environments where additional cooling is not feasible, necessitating the use of processors that generate minimal heat and consume low power. As a result, despite their performance benefits, GPUs

Communicated by Jihye Park.

✉ Heoncheol Lee
heonc.lee@gmail.com

[1] School of Electronic Engineering, Kumoh National Institute of Technology, Gumi, South Korea

[2] Department of IT Convergence Engineering, School of Electronic Engineering, Kumoh National Institute of Technology, Gumi, South Korea

[3] PGM R&D Lab, LIGNEX1, Seongnam 13488, Republic of Korea

Published online: 11 April 2025

Table 1 Related works to MCO for target tracking

Related works	FPGA-based parallelization	Parallelization part	Missile Utilization
[11–15]	×	×	×
[16–18]	○	Generating Samples	×
[19]	×	×	○
ours	○	Generating Samples, Calculation likelihood	○

are unsuitable for such applications [8]. In contrast, Field-Programmable Gate Arrays (FPGAs) offer lower power consumption and minimal heat generation, enabling efficient operation without the need for active cooling. This makes them ideal for embedded systems where passive or minimal cooling is required. While TPUs have gained traction as AI accelerators, they still generate substantial heat due to high-density computation and sustained workloads [9, 10]. Moreover, the requirement for cooling systems to manage this heat renders them unsuitable for our guided weapon systems, prompting us to adopt FPGA-based solutions instead.

Table 1 summarizes related studies on MCO for target tracking. While several studies applied non-accelerated MCO across various domains [11–14], others explored parallelization of the sampling generation process using MCO in different applications [16–18]. MCO techniques have also been specifically applied to missile guidance and tracking [19]. Therefore, this study proposes an FPGA-based parallelization of MCO tailored for ballistic target tracking, addressing both the computational and power efficiency challenges.

2 Problem Description

2.1 Sampling-Based Ballistic Target Tracking

The primary objective of this study is to develop an object-tracking algorithm capable of accurately estimating the current state of a target in real-time. The trajectory of a ballistic missile is modeled to evaluate the accuracy of this estimation through likelihood calculations, while also profiling the computational demands involved. Unlike simple free-fall motion, a ballistic missile during atmospheric reentry is significantly influenced by aerodynamic drag and gravitational forces, both of which critically alter its trajectory. In this study, the ballistic missile is modeled as a point mass within a Cartesian coordinate system. Additionally, symmetrical aerodynamic characteristics of the missile are assumed, allowing the use of simplified nonlinear dynamic equations to represent its motion as follows:

$$\dot{x} = V\cos\gamma\cos\sigma,\ \ \dot{y} = V\cos\gamma\sin\sigma,\ \ \dot{z} = -V\sin\gamma, \quad (1)$$

$$\begin{aligned}\dot{\sigma} &= \frac{F_l \sin\varepsilon}{mV\cos\gamma},\\ \dot{V} &= \frac{F_t - F_d - mg\sin\gamma}{m},\\ \dot{\gamma} &= \frac{F_l\cos\varepsilon - mg\cos\gamma}{mV},\end{aligned} \quad (2)$$

$$F_d = \frac{1}{2}\rho V^2 \cdot C_d \cdot S,\ \ F_l = \frac{1}{2}\rho V^2 \cdot C_l \cdot S \quad (3)$$

where x, y, and z represent the target's position in the Cartesian coordinate system, while V denotes its velocity. The variables σ and γ correspond to the heading angle and the flight path angle, respectively. The gravitational constant is represented by g, and m is the mass of the object. The forces F_t, F_d, and F_l denote thrust, drag, and lift, respectively. Air density is indicated by ρ, the lift coefficient denoted by $C_{l,}$, and the drag coefficient by C_d. The reference area is denoted by S, and ε i signifies the direction of lift generation.

This study focuses on the atmospheric reentry phase of a ballistic object. During this phase, the propulsion system is typically no longer active, and as a result, thrust F_t is assumed to be zero and the mass m of the ballistic object is also considered constant. Furthermore, due to the symmetric configuration of the ballistic object and negligible maneuvering, the lift force F_l is not generated during atmospheric reentry. Consequently, the dynamic Eq. (2) is simplified as follows:

$$\begin{aligned}\dot{\sigma} &= 0,\\ \dot{V} &= -\frac{F_d}{m} - g\sin\gamma,\\ \dot{\gamma} &= -\frac{g\cos\gamma}{V},\end{aligned} \quad (4)$$

And the trajectory of the ballistic object is conceptually represented in the Cartesian coordinate system, as illustrated in Fig. 1.

2.2 Movement and Motion Model of the Target

Accurate modeling of target motion is a critical step in the tracking process, and various methodologies are available for this purpose. In this study, the Singer model was employed to model the target's motion [20, 21]. The Singer model assumes that the target experiences zero-mean acceleration and represents this behavior as a first-order stationary Markov process. This model effectively captures the

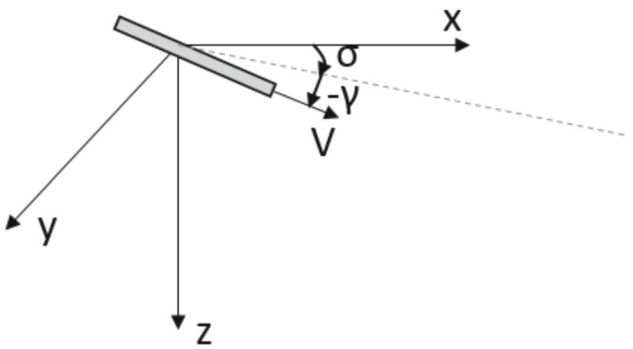


Fig. 1 Trajectory of the ballistic object in the Cartesian coordinate system

temporal evolution of the target's state, which can be mathematically described as follows:

$$\dot{x} = Ux + J\omega, \tag{5}$$

$$U = \begin{bmatrix} 0_3 & I_3 & 0_3 \\ 0_3 & 0_3 & I_3 \\ 0_3 & 0_3 & \frac{-1}{\tau} I_3 \end{bmatrix}, \quad J = \begin{bmatrix} 0_3 \\ 0_3 \\ I_3 \end{bmatrix}, \tag{6}$$

where $\dot{x}$ is the state of the tracked target, and ω represents white Gaussian noise with a mean of zero and a time constant τ.The matrices U and J include the 3×3 identity matrix I_3, and τ is the mobilization constant. The discrete equations for ω are expressed as follows:

$$x_k = \Phi_{k-1} x_{(k-1)} + \omega_{(k-1)}, \quad \omega_{(k-1)} \sim N(0, C_k), \tag{7}$$

$$\Phi_k \simeq I + U\Delta t, \tag{8}$$

$$C_k \simeq P_\omega C_0 = P_\omega \begin{bmatrix} \frac{\Delta t^5}{20} I_3 & \frac{\Delta t^4}{8} I_3 & \frac{\Delta t^3}{6} I_3 \\ \frac{\Delta t^4}{8} I_3 & \frac{\Delta t^3}{3} I_3 & \frac{\Delta t^4}{2} I_3 \\ \frac{\Delta t^6}{6} I_3 & \frac{\Delta t^2}{2} I_3 & \Delta t I_3 \end{bmatrix}, \tag{9}$$

where Φ_k denotes the state transition matrix, and Δt represents the sampling interval. The covariance C_k is constructed using P_w, the power spectral density, and C_0, the white-noise jerk model. The increase in acceleration over time was quantified using the integral of the jerk over a given period. Here, x is defined as follows:

$$x = \begin{bmatrix} P^T & V^T & A^T \end{bmatrix}^T, \tag{10}$$

$$P = \begin{bmatrix} x & y & z \end{bmatrix}^T, \tag{11}$$

where P, V, and$A$ are the position, velocity, and acceleration, respectively, in the Cartesian coordinate system. Vectors $\begin{bmatrix} x & y & z \end{bmatrix}$ represent the position of the target in a three-dimensional space. Based on this definition, we defined the measured data. We assumed that radar was used to measure the target's altitude, attitude, and distance. These measurements can fluctuate based on the position of the target relative to the radar as described below.

$$\begin{bmatrix} x_r & y_r & z_r \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T - \begin{bmatrix} x_m & y_m & z_m \end{bmatrix}^T, \tag{12}$$

where x_r, y_r, and z_r denote the relative positions of the target and the radar, respectively, and x_m, y_m, and z_m denote the coordinates of the radar. Consequently, the two bearing angles q_ψ and q_θ and relative distance q_R can be obtained by

$$\begin{bmatrix} q_\theta \\ q_\psi \\ q_R \end{bmatrix} = \begin{bmatrix} \tan^{-1}\left(\frac{z_r}{\sqrt{x_r^2 + y_r^2}}\right) + n_{G,\theta} + n_\theta \\ \tan^{-1}\left(\frac{y_r}{x_r}\right) + n_{G,\psi} + n_\psi \\ \sqrt{x_r^2 + y_r^2 + z_r^2} + n_R \end{bmatrix} \tag{13}$$

The above equation includes the noise associated with the observed position relative to the actual position of the target. Measurements include noise, with measurement errors represented by Gaussian noise from the radar receivers n_θ, n_ψ, and n_R. Additionally, non-Gaussian noise $n_{G,\theta}$ and $n_{G,\psi}$ are included to reflect radar glint noise [22].

2.3 Real-Time Problem

The ability to rapidly process data is critical in target-tracking systems, particularly for high-speed targets, such as ballistic missiles, where real-time trajectory prediction and accuracy are paramount. The precision of target tracking is the most significant factor in enhancing interception rates. To improve prediction accuracy, we employed a Monte Carlo Optimization (MCO) algorithm to probabilistically analyze the target's potential trajectories.

During this process, information extraction and numerical computations on the samples are performed iteratively for each cycle. While increasing the number of samples enhances accuracy, it also significantly escalates the computational load. The resulting increase in computations makes real-time tracking increasingly challenging. To address this problem, previous research has utilized GPU-based parallelization to accelerate the process [4, 23]. However, due to the high power consumption of GPUs, this approach is not feasible in power-constrained environments. Consequently, we propose an FPGA-based acceleration algorithm that offers enhanced computational performance with lower power consumption.

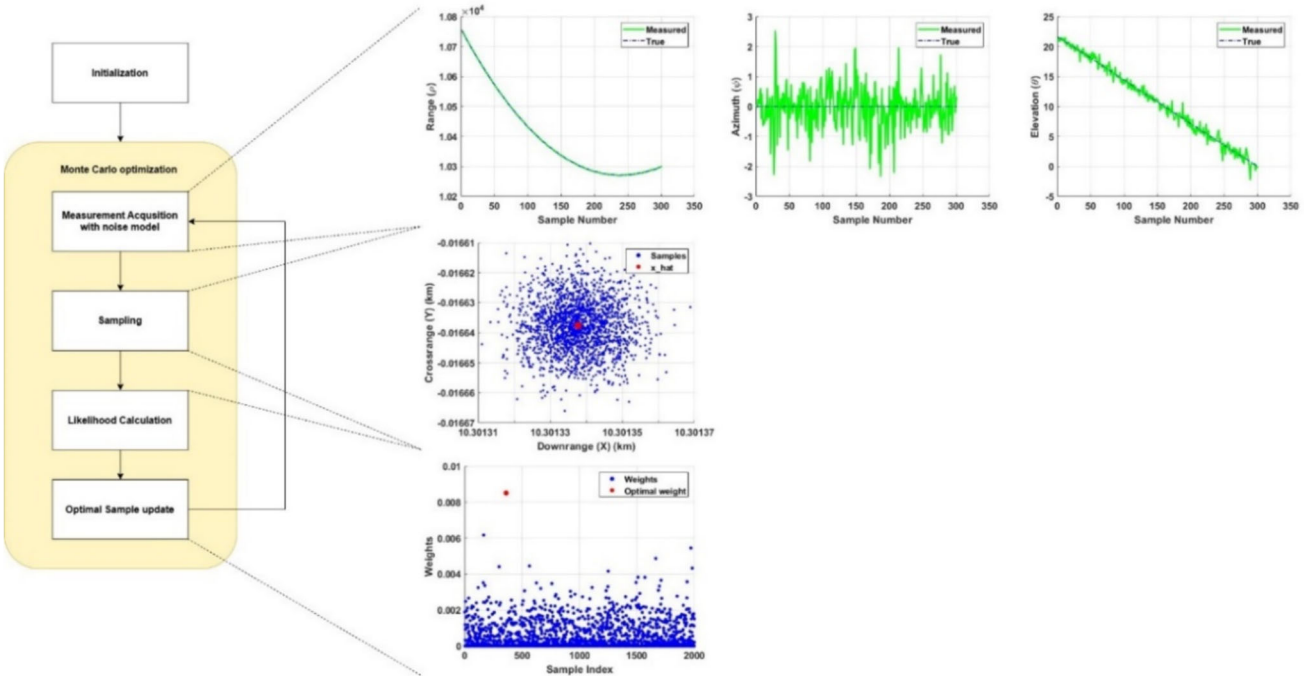


Fig. 2 The process of Monte Carlo optimization (MCO) for ballistic target tracking

3 FPGA-Based Pipelining Technique for MCO

Figure 2 illustrates the Monte Carlo Optimization (MCO) process for ballistic target tracking. To implement the MCO technique, we developed a simulation model designed to accurately track high-speed ballistic targets. In real-world scenarios, noise introduced during target tracking can significantly impact the precision of position estimates. The MCO method addresses this by enhancing state tracking accuracy through the generation of samples that represent random positional information. In the Monte Carlo method, each sample corresponds to a specific point in space, encapsulating data on the ballistic target's location. These samples are generated with inherent uncertainty, modeled through randomly allocated numbers that simulate the noise encountered during the tracking process. The MCO algorithm then applies these samples to a measurement function to derive optimal estimates, selecting the sample with the highest calculated value as the optimal estimate. This estimation process is iterative, with each cycle involving the application of new random numbers to refine the estimated values. The process continues until the predefined number of iterations is reached, ensuring that the MCO method efficiently tracks the trajectory of ballistic targets. The accuracy of this approach is validated by comparing the estimates generated by the MCO algorithm against actual measurements from the ballistic missile target model.

3.1 Computation Time Profiling

In the Monte Carlo Optimization (MCO) method, increasing the number of samples generally enhances accuracy; however, it also significantly increases the computational load and extends execution time. To address this challenge, the execution time of each operation within the algorithm was meticulously measured to identify the most time-consuming processes. As depicted in Fig. 3, the sample size was set to 2000, and the computation time for each algorithmic component was recorded using only a CPU. The FPGA implementation was conducted on a Zynq UltraScale + ZCU104 board.

As shown in Fig. 3, the computational times for the optimal sample update and initialization operations were relatively minor, constituting only 0.483% and 8.41% of the total time, respectively. In contrast, sampling and likelihood calculations consumed the majority of the time, accounting for 64.197% and 35.31%, respectively. Based on these findings, this study concentrated on accelerating the algorithm by synthesizing the sampling and likelihood calculation components into IP cores using Xilinx Vitis HLS, which were then implemented on the FPGA.

3.2 Parallelized Sampling

To reduce the computation time of the sampling process, the motion model was analyzed for possible parallelization as

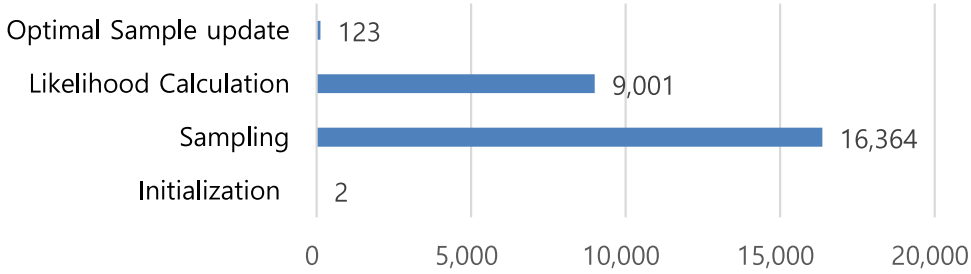


Fig. 3 Computation time profiling

follows:

$$S_{update} = (H \times S_{hat}) + \left(H \times \sqrt{C_k} \times rnd\right) \tag{14}$$

where H is the state transition matrix for the target. S_{hat} is a 9×1 matrix that holds three-dimensional information about the initial samples' position, velocity, and acceleration. C_k is the error covariance matrix representing noises. rnd is a random number that follows a zero-mean normal distribution with a standard deviation of one. In this method, noise is generated in the processing system (PS) part and fed as data input through the AXI stream interface, which is designed to be processed internally by the IP. This acceleration process is shown in Table 2.

The state transition matrix H and the covariance matrix for noise Ck, established in the initial modeling, are initialized internally within the IP. This approach reduces the overhead associated with data input and output.

Figure 4a illustrates the conventional operating method, where all operations from Compute1 are executed sequentially to process 1000 samples. This sequential processing, however, requires a substantial amount of time. To address this inefficiency, we propose parallelizing operations using a pipelining approach. Due to the dependencies among the computed Eq. (14), parallelization within a single sample was not feasible. Instead, we implemented parallelization between samples.

As depicted in Fig. 4b, the operations for sample $n = 0$ are initiated first, followed by the operations for sample $n = 1$ after a one-clock-cycle delay. This strategy of initiating the operations for the next sample at every clock cycle enables parallel processing across samples without compromising intra-task dependencies. As a result, this approach significantly improves processing speed. Figure 4a shows the conventional operating method. In this method, all operations from Compute1 were performed sequentially to process 1000 samples. However, sequential processing requires considerable time. Therefore, we propose the parallelization of operations using the pipelining method. However, because of the dependencies among the calculated Eq. (14), the parallelization of operations within a single sample was impossible. Instead, we parallelized the operations between samples. As shown in Fig. 4b, the operations of sample $n = 0$ begin first, followed by those of sample $n = 1$ after one clock cycle. The approach of starting the operations of the next sample at every clock cycle allows parallelization between samples without compromising intra-task dependencies, resulting in a significant improvement in processing speed.

Calculations involving decimal precision are crucial in ballistic missile tracking, necessitating the use of the double data type. During data transfer from the Processing System (PS) to the Programmable Logic (PL), floating-point double data is converted into fixed-point 'uint_64' format to ensure precise data transmission. Consequently, an additional conversion process is required to revert the data type to double during data retrieval. Figure 5a illustrates the process of decoding the data and writing the values to the buffer. Similar to the approach in Fig. 4b, the data reading, decoding, and buffer writing operations were pipelined, allowing concurrent processing of multiple samples. The results of this pipelining process are shown in Fig. 5b. When writing the results, the data are converted from double to uint_64, reversing the earlier reading process.

To enhance processing speed, each sample completes its calculations and writes the data sequentially, as depicted in Fig. 6a. Since data writing occurs only after the completion of a sample, the volume of data processed simultaneously is limited. To address this, the UNROLL directive was employed to enable parallel data access, significantly reducing processing time, as shown in Fig. 6b.

3.3 Parallelized Likelihood Calculation

The likelihood calculation is the second most time-consuming part of the MCO method, following the sampling part. These functions can be divided into two main categories. The information obtained from the samples was used to estimate the target's distance, angle, and rotation angle as follows:

$$\text{Range}[n] = \sqrt{S_1^2[n] + S_2^2[n] + S_3^2[n]}, \tag{15}$$

$$\theta[n] = -\arctan\left(\frac{S_3[n]}{\sqrt{S_1^2[n] + S_2^2[n]}}\right) \times \frac{180}{\pi}, \tag{16}$$

Table 2 Pseudo-code for parallelized sampling

Algorithm 1 Parallelized Sampling in MCO

```
N ← 1,000                                                    ▷ Number of samples
Define matrix H [9][9] and √C_k[9][9] as given
procedure Sampling (in_rand, in S_hat, S_update )
  for i ← 0 to 8 do
    #pragma HLS UNROLL
    stream ← in_S_hat.read()
    S_hat[i] ← uint64 to double(stream.data)
  end
 for i ← 0 to N − 1 do
   for j ← 0 to 8 do
     #pragma HLS PIPELINE
     stream ← in_rand.read()
     rnd[i][ j] ← uint64 to double(stream.data)
   end
 end
 for n ← 0 to N − 1 do
   #pragma HLS PIPELINE
   Perform matrix multiplication of √C_k with rnd[n] to get √C_k _rnd
   Perform matrix multiplication of H with √C_k rnd to get H _√C_k_ rnd
   Add H multiplied by S_hat to H _√C_k _rnd to get final result[n]
   for i ← 0 to 8 do
     #pragma HLS UNROLL
     stream.data ← double to uint64(result[n][i])
     stream.last ← (n== N − 1)and(i == 8)?1 : 0
   S_update.write(stream)
   end
  end
end
```

$$\psi[n] = \arctan\left(\frac{S_2[n]}{S_1[n]}\right) \times \frac{180}{\pi}, \tag{17}$$

where the variables S_1 [n], S_2 [n], and S_3 [n] represent the x-, y-, and z-coordinate information of samples taken from random positions, and n is the number of samples, allowing calculations for each sample.

With these estimates, we can input them into the likelihood function to calculate the likelihoods for distance, angle, and rotation angle as follows:

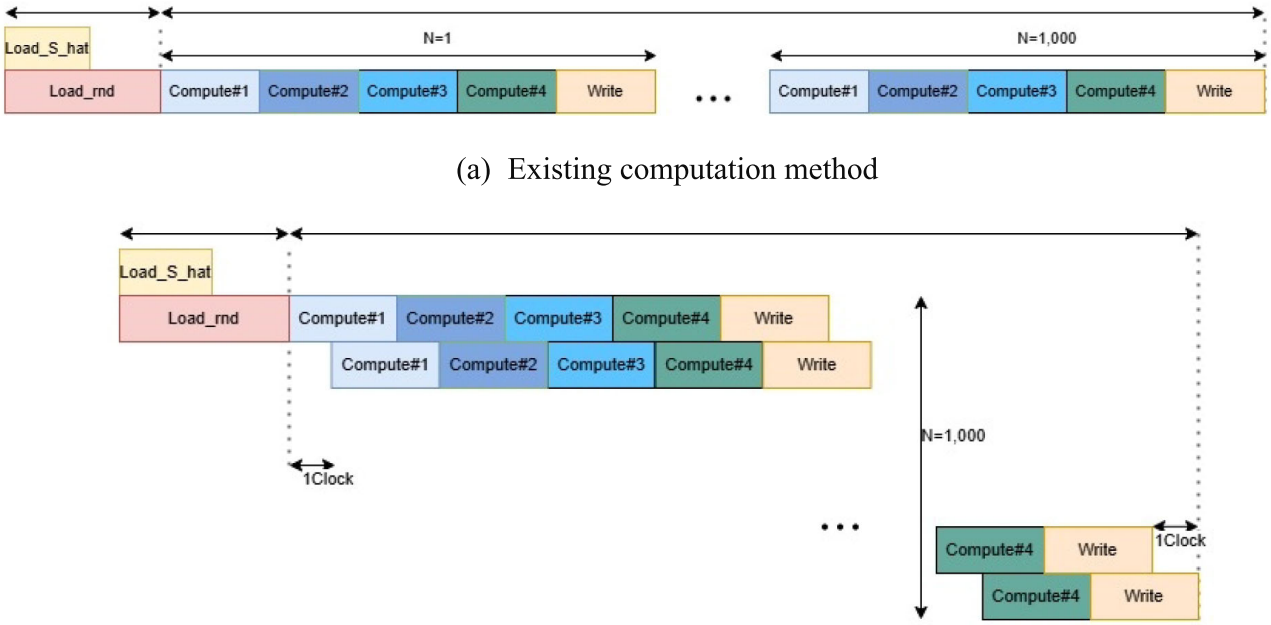
$$P_{\text{Range}} = \frac{1}{\sqrt{2\pi} \times \sigma_\rho} \times \exp\left(-\frac{(\text{mean}_R - \text{Range})^2}{(2 \times \sigma_\rho)^2}\right), \tag{18}$$

where σ_ρ denotes the measurement noise, and mean_R represents the acquired measurement of the target's distance. In this study, σ_ρ was set to 1. P_{Range} refers to the estimated distance obtained during the predicted measurement phase.

The likelihood functions and internal computations for the angles and rotation angles are defined by

$$\text{temp1} = (1 - ep) \times \left(\frac{1}{\sqrt{2\pi\left(\sigma_{\theta,\psi}^2 + \left(\frac{\sigma_{g1}}{\text{Range}^2}\right)\right)}}\right), \tag{19}$$

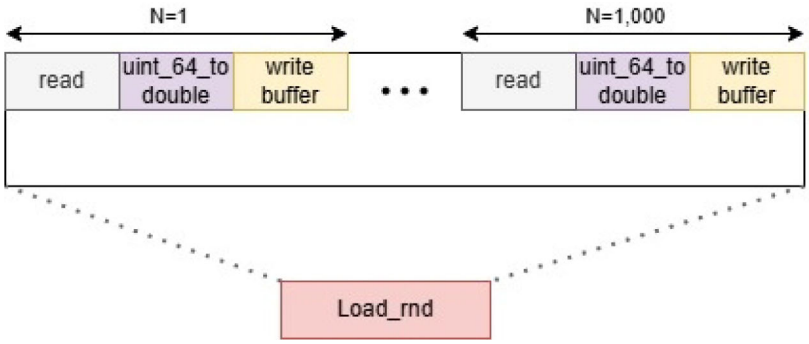
$$\text{temp2} = \exp\left(-\frac{(\text{mean}_{\theta,\psi} - \theta,\ \psi)^2}{2\left(\frac{(\sigma_{\theta,\psi} + \sigma_{g1})^2}{\text{Range}^2}\right)}\right), \tag{20}$$



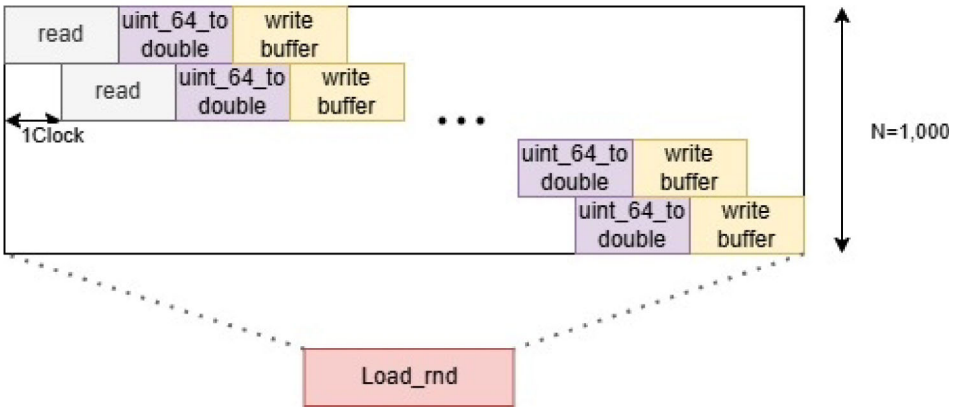
(a) Existing computation method

(b) Proposed computation method

Fig. 4 Results of pipelining the computations between samples



(a) Existing computation methods



(b) Proposed computation method

Fig. 5 Results of pipelining the data read in sampling

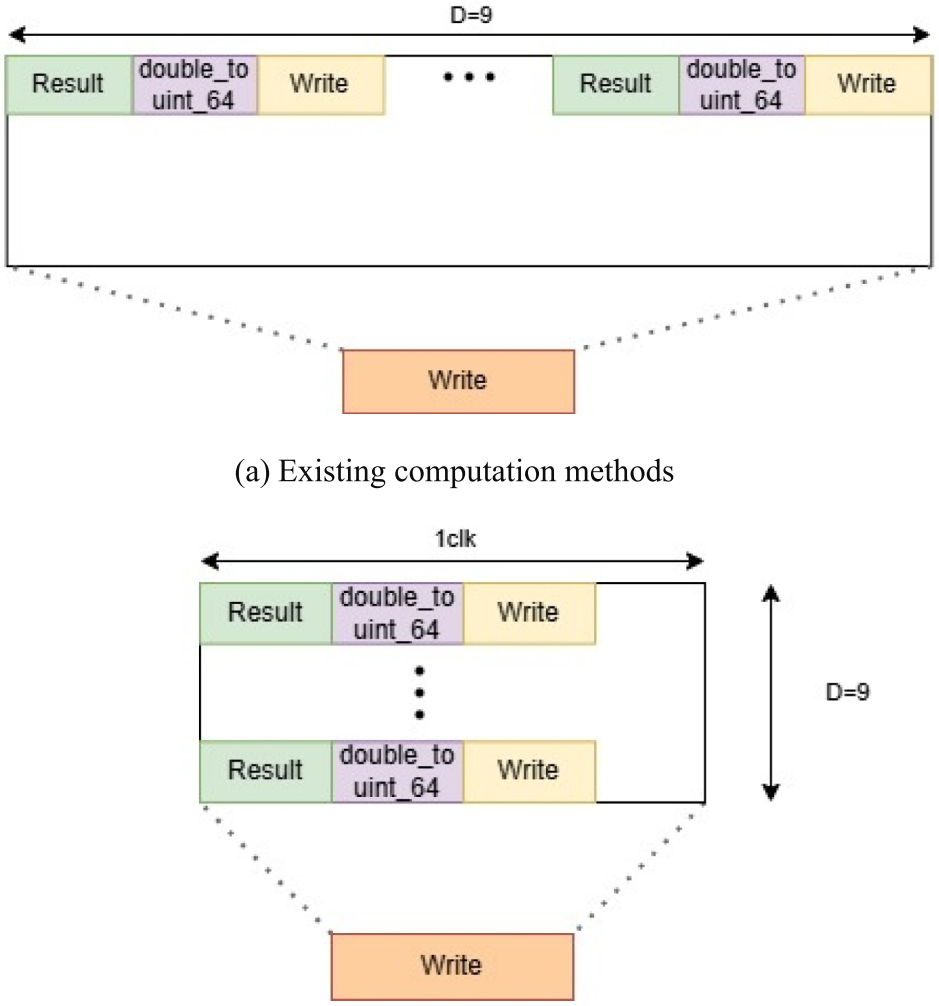


Fig. 6 Results of unrolling the data read in sampling

(a) Existing computation methods

(b) Proposed computation method

$$\text{temp3} = (\text{ep}) \times \left(\frac{1}{\sqrt{2\pi\left(\sigma_{\theta,\psi}^2 + \left(\frac{\sigma_{g2}}{\text{Range}^2}\right)\right)}} \right), \tag{21}$$

$$\text{temp4} = \exp\left(-\frac{\left(\text{mean}_{\theta,\psi} - \theta,\ \psi\right)^2}{2\left(\frac{\left(\sigma+\sigma_{g2}\right)^2}{\text{Range}^2}\right)} \right), \tag{22}$$

$$P_\psi = \text{temp1} \times \text{temp2} + \text{temp3} \times \text{temp4}, \tag{23}$$

$$P_\theta = \text{temp1} \times \text{temp2} + \text{temp3} \times \text{temp4}, \tag{24}$$

where $\sigma_{\theta,\psi}$ represents the measurement noise, and the $\text{mean}_{\theta,\psi}$ represents the measurement mean; both vary depending on the type of measurement being conducted. σ_θ was set to 0.1 for the angle measurement noise, and σ_ψ was also set to 0.1 for the rotation angle measurement noise. σ_{g1} and σ_{g2}, which occur during estimation, were set to 0.5 and 1, respectively. ep is the probability of noise occurring owing to the glint.

Using the likelihoods P_{Range}, P_ψ, and P_θ, we can update the weight W for N samples by multiplying these likelihoods together as follows:

$$W = P_{\text{Range}} \times P_\psi \times P_\theta, \tag{25}$$

where P_{Range} represents the likelihood of the distance, and P_ψ and P_θ represent the likelihoods of the two bearing angles. These values are multiplied to obtain the overall likelihood, W, representing the accuracy of the estimate compared with the actual measurement. The W value of each N sample was calculated, and the highest sample value was used as the optimal sample. This method demonstrates that the more samples available for evaluation, the more accurate the target estimation. However, this method increases the computational load;

Table 3 Pseudo-code for parallelized likelihood calculation

Algorithm 2 Likelihood Calculation

```
1: N ← 1,000                                                    ▷ Number of Samples
2: P I ← 3.14159
3: sig_rho ← 1.0
4: sig _theta ← −0.1
5: sig_psi ← 0.1
6: ep ← 0.3
7: sig _g1 ← 0.5
8: sig _g2 ← 1.0
9: r2d ← 180/P I
10: procedure Likelihood Calculation (in_S_m, in_S_bar, out_W)
11:     for i ← 0 to 2 do
12:         stream ← in_S_m. read ()
13:         S_m[i] ← u64 to double(stream.data)
14:     end
15:     for i ← 0 to N − 1 do
16:         for j ← 0 to 2 do
17:             #pragma HLS PIPELINE
18:             stream ← in_S_bar. read ()
19:             S_bar[j][i] ← u64 to double(stream.data)
20:         end
21:     end
22:     for i ← 0 to N − 1 do
23:         #pragma HLS PIPELINE
24:         Calculate Eρ[i]
25:         Calculate E0[i]
26:         Calculate Eψ[i]
27:     end
28:     for i ← 0 to N − 1 do
29:         #pragma HLS PIPELINE
30:         Calculate Pρ[i]
31:         Calculate P0[i]
32:         Calculate Pψ[i]
33:         Compute weight W[i]
34:     end f
35:     for i ← 0 to N − 1 do
36:         #pragma HLS PIPELINE
37:         stream.data ← double to uint64(W[i])
38:         stream. last ← (i == N − 1)?1: 0
39:         out_W. write(stream)
40:     end
41: end
```

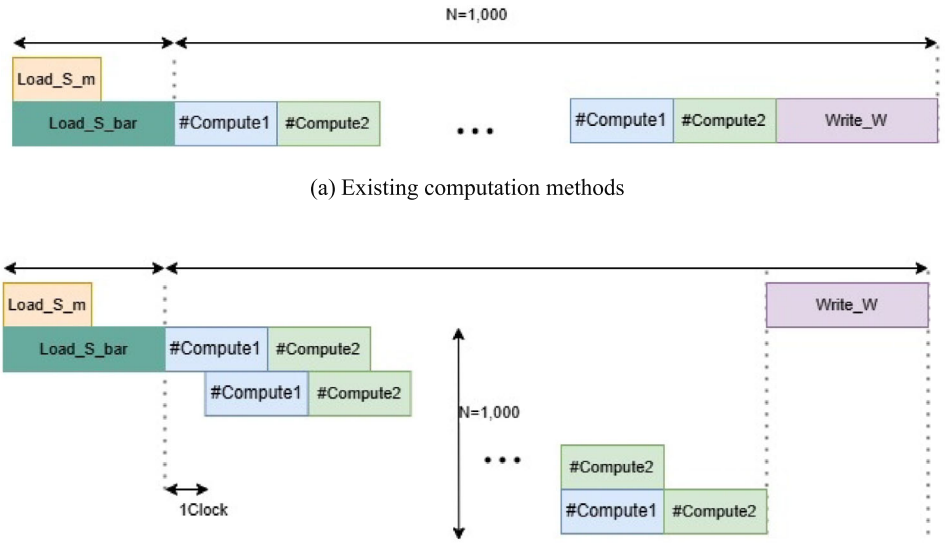


Fig. 7 Results of pipelining the likelihood function

(a) Existing computation methods

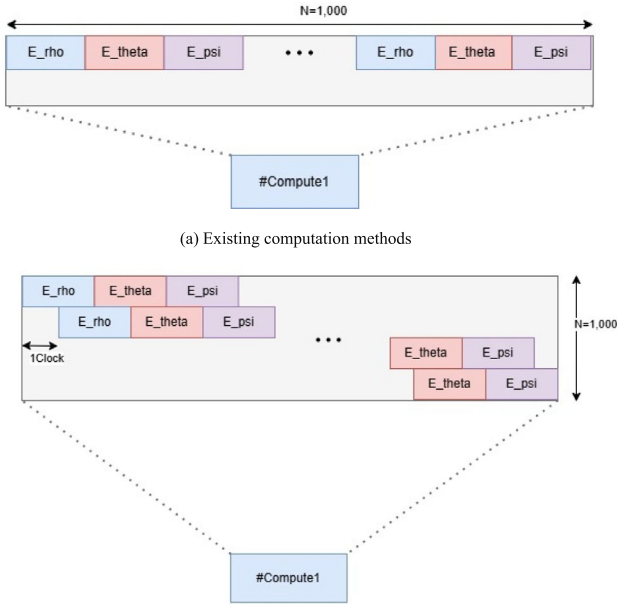
(b) Proposed computation method

therefore, we designed an IP that calculates the likelihood in parallel using Vitis HLS.

In Table 3, the noise values and initial settings such as 'ep' were first declared in the BRAM within the IP to reduce the overhead of inputs and outputs. The design reads the actual measurement value S_m and the estimate value S_{bar} obtained through sampling as inputs and outputs the value of the weights. The speed of each read and write was enhanced using the pipelining method described above. The estimated distance, angle, and rotation angle were subsequently computed. The likelihood calculation depends on these estimated values and can only be performed after they are determined. However, within the loops responsible for calculating the estimates and likelihoods, there is no interdependency between the computations of distance, angle, and rotation angle. As a result, we were able to implement parallelization by pipelining these calculations within a loop, free of dependencies.

In Fig. 7a, Compute1 calculates the estimates for distance, angle, and rotation angle, while Compute2 computes the likelihood for these parameters to determine the weight W. Compute2 bases its likelihood calculations on the values obtained from Compute1. As shown in Fig. 7b, parallelization between samples was implemented, allowing the pipelines to operate independently and avoid interdependencies.

The computations within Compute1 and Compute2 are designed to be sequential, as depicted in Fig. 8a, where operations are executed in order. However, since there is no dependency between these operations, pipelining was applied. As illustrated in Fig. 8b, the computation of E_rho begins immediately, followed by the computation of E_theta.



(a) Existing computation methods

(b) Proposed computation method

Fig. 8 Results of pipelining Compute1

Pipelining occurs at one-clock-cycle intervals, enabling parallel processing for each sample. This approach enhances parallelism and improves performance by enabling pipelined execution within the Compute1 and Compute2 operations for each sample.

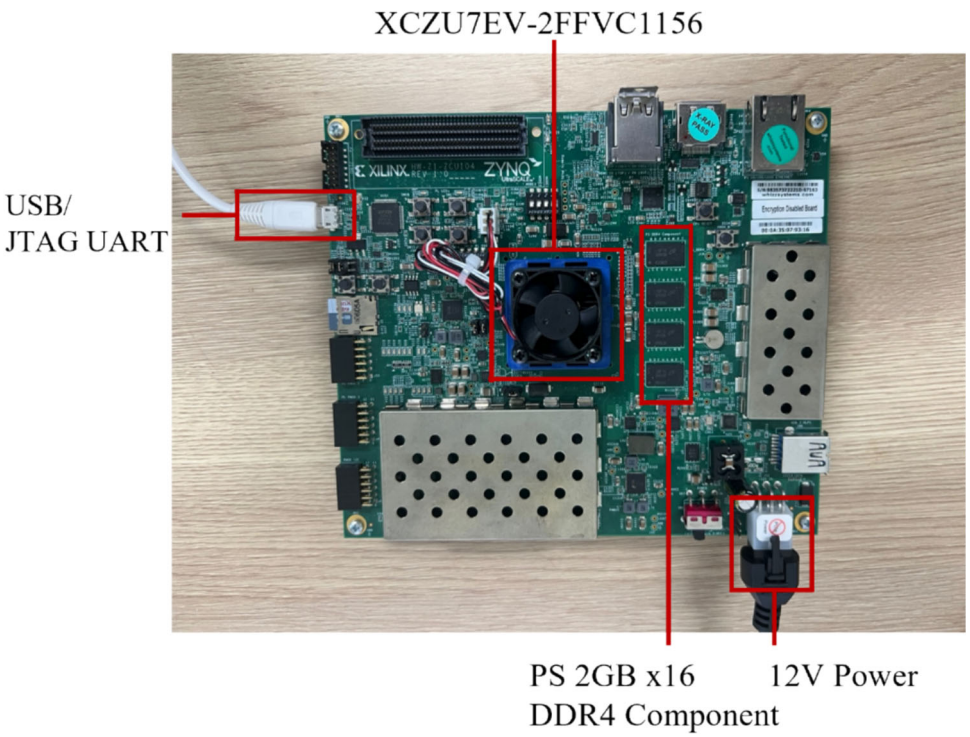


Fig. 9 Board used in the experiment

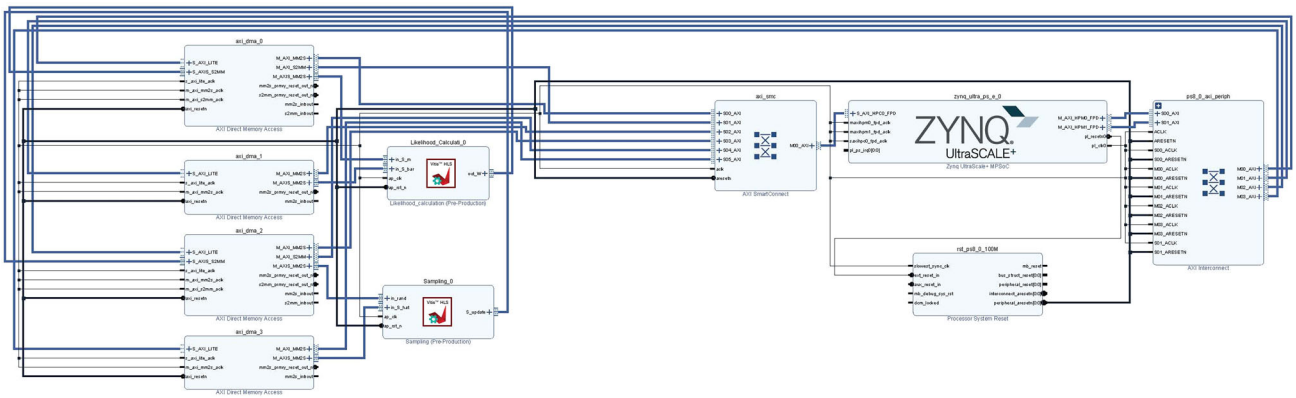
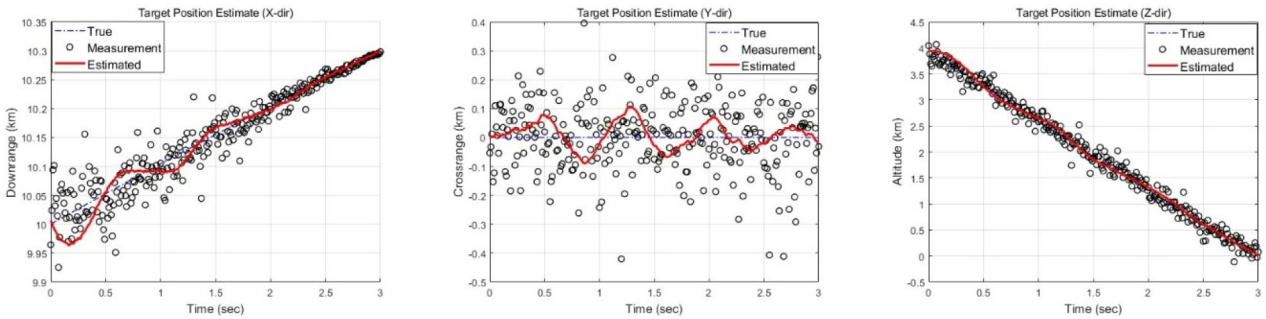


Fig. 10 Implementation results of the proposed method



(a) Estimated target downrange (X-dir)

(b) Estimated target crossrange (Y-dir)

(c) Estimated target altitude (Z-dir)

Fig. 11 Target tracking results by the proposed parallelized MCO on the FPGA when $N = 2000$

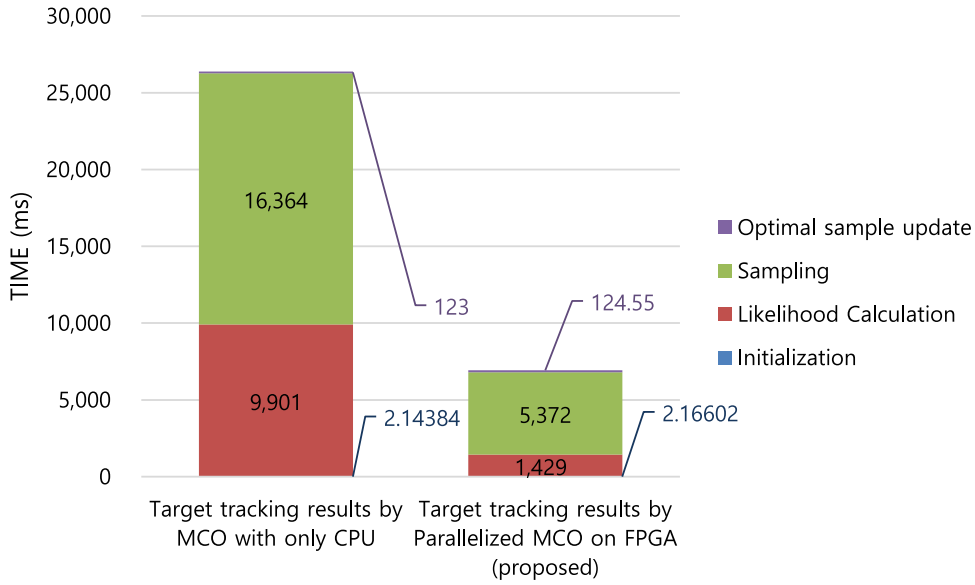


Fig. 12 Comparison of computation times of logical blocks

4 Evaluation Results

4.1 Target Board

For Methods 1 and 2, the designed hardware IPs were implemented on an UltraScale + ZCU104 (XCZU7EV-2FFVC1156) board as depicted in Fig. 9. The ZU7EV device integrates a quad-core ARM® Cortex™-A53 application processor, a dual-core Cortex-R5 real-time processor, a Mali™-400 MP2 GPU, and 16 nm FinFET + programmable logic (PL). The platform was designed using Vivado 2022.1.

In the design shown in Fig. 10, communication between the hardware's Processing System (PS) and Programmable Logic (PL) is established through the AXI stream protocol, utilizing AXI Direct Memory Access (DMA) for data transfer. This design incorporates two IPs and four DMAs, with the AXI SmartConnect and Interconnect IP provided by Vivado used for mapping between the master and slave nodes. The developed platform was tested on a PC using Vitis 2022.1.

4.2 Comparison of Ballistic Target Tracking Results

The glint noise model was represented as follows:

$$p = (1 - \tau)p_{G1} + p_{G2}, \tag{26}$$

where τ represents the probability of glint, P_{G1} is modeled as a Gaussian distribution $p_{G1} \sim N(0, 0.1^2)$, and P_{G2} follows a Gaussian distribution $p_{G2} \sim N(0, 1^2)$. This model effectively simulates the glint noise in the tracking system. The tracking motion model employed the Singer model, as defined in Eq. (6), while the measurement model was derived using Eq. (13). The radar was assumed to be fixed at a ground location, with the ballistic target moving at high speeds, influenced by both gravity and aerodynamic drag. Consequently, the velocity of the ballistic target varied over the course of the simulation.

Figure 11 presents the results of trajectory and state tracking using a sample size of $N = 2000$ for the MCO

Table 4 Comparison of computation times on target tracking results

N(Sample)	CPU (ms)	Proposed (ms)	Acceleration (×)
1000	15,201	7888	1.93
2000	30,303	11,714	2.59
3000	43,621	15,739	2.77
4000	56,942	19,765	2.88
5000	70,320	23,794	2.955
6000	82,445	25,761	3.20
7000	95,903	29,778	3.22
8000	10,9345	33,771	3.24
9000	12,2861	37,768	3.25
10,000	13,6329	41,774	3.26

Table 5 Resource use

Resource	Utilization	Available	Utilization%
LUT	125,714	230,400	54.56
LUTRAM	4185	101,760	4.11
FF	86,307	460,800	18.73
BRAM	118	312	37.82
DSP	663	1728	38.37
BUFG	2	544	0.3

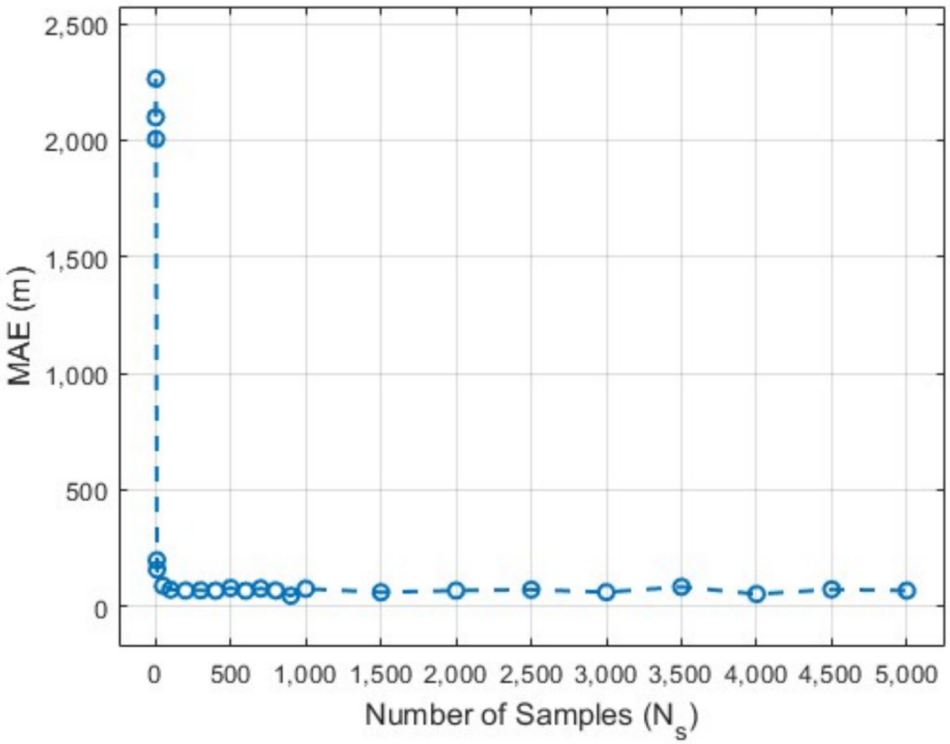


Fig. 13 Trade-off between the number of samples (N_s) and MAE (m)

algorithm implemented on an FPGA. In the graph, the blue line represents the actual target trajectory, circles indicate the measured values, and the red line denotes the estimated values obtained using the FPGA-accelerated MCO algorithm. The results confirm that the FPGA-based implementation effectively performs target estimation with accuracy comparable to the original CPU-based algorithm. To evaluate estimation accuracy, we use the Mean Absolute Error (MAE), which measures the average absolute difference between estimated and true positions across the x, y, and z coordinates. A lower MAE indicates a trajectory that more closely follows the actual target path. In this comparison, the FPGA-based implementation achieves an MAE of 68.958 m, while the CPU-based version results in 72.316 m. The small 3.358 m difference demonstrates that the proposed acceleration method preserves the original algorithm's estimation performance while running on hardware.

4.3 Analysis of Computation Times of Logical Blocks

The comparison of computational times by operation, as shown in Fig. 12, reveals that the sampling process is approximately 3.05 times faster, and the likelihood calculation is about 6.3 times faster when utilizing the parallelized MCO on the FPGA (proposed) approach compared to the MCO executed solely on a CPU. Additionally, as shown in Table 4, when comparing computation speeds across different sample sizes, it becomes evident that the disparity in computation speed between the CPU-only MCO and the parallelized MCO on the FPGA (proposed) approach increases as the number of samples grows.

The comparison of computational times by operation, as shown in Fig. 12, indicates that the sampling process is approximately 3.05 times faster, and the likelihood calculation is about 6.3 times faster when using the proposed parallelized MCO on the FPGA approach, compared to the MCO executed solely on a CPU. Moreover, as demonstrated in Table 4, the disparity in computation speed between the CPU-only MCO and the proposed parallelized MCO on the FPGA approach becomes increasingly pronounced as the number of samples increases.

Figure 13 shows how the MAE varies with the number of samples N. When N is very small, the MAE remains high, indicating that the estimation is less accurate. However, as N increases particularly around 500 samples, the MAE rapidly approaches near-zero levels, demonstrating that a larger sample size significantly improves estimation accuracy. In more extreme noise environments, this convergence would require even more samples to maintain similar accuracy, significantly extending the time and computational cost needed for estimation.

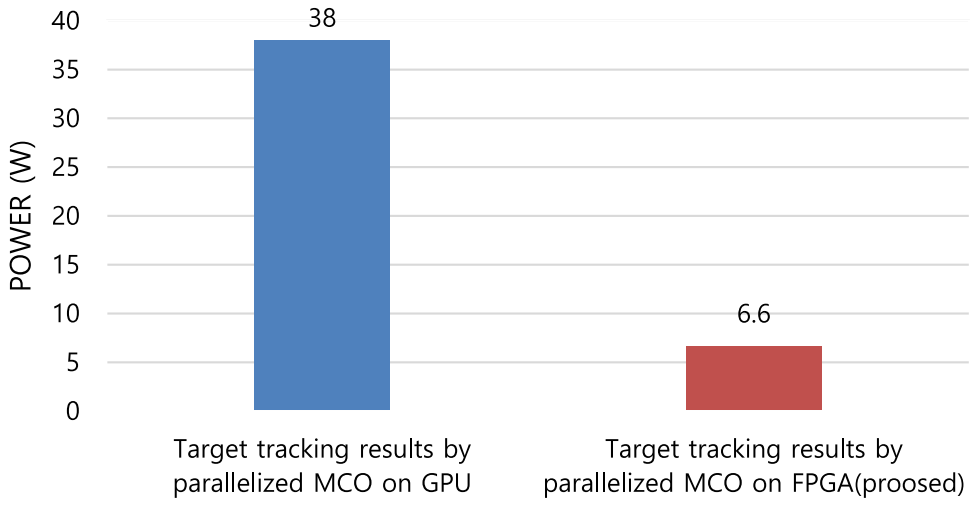
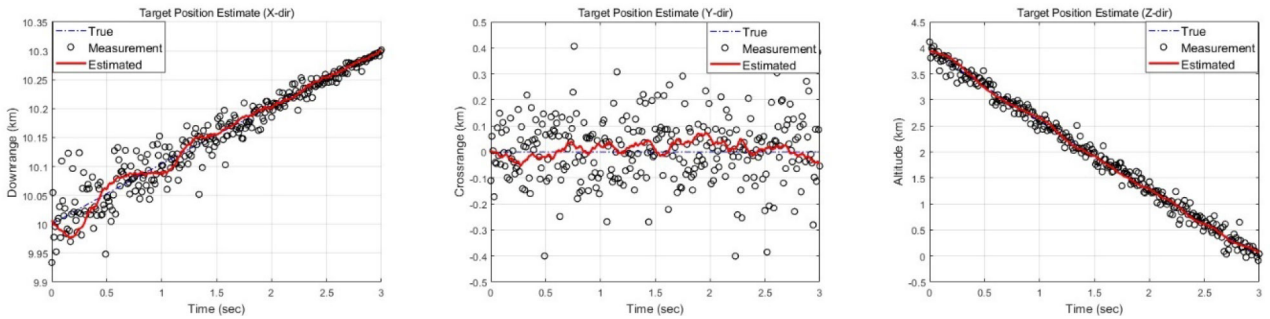


Fig. 14 Comparison of power use



(a) Estimated target downrange (X-dir)

(b) Estimated target crossrange (Y-dir)

(c) Estimated target altitude (Z-dir)

Fig. 15 Target tracking results by parallelized PSO on the FPGA when $N = 2000$ and $epoch = 5$

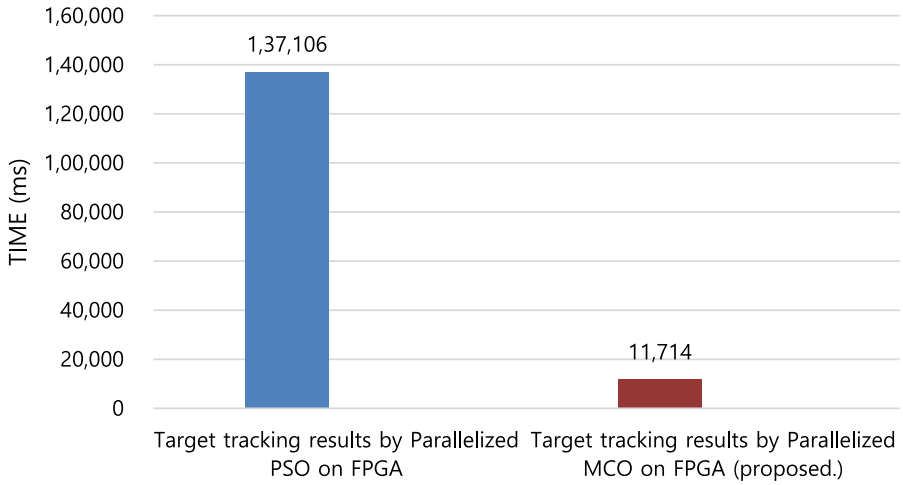


Fig. 16 Comparison of the computation times of the parallelized PSO on the FPGA [25] and the proposed method

4.4 Resource Use Analysis

Both IP cores developed using Methods 1 and 2 were designed with a reference of $N = 1000$ samples, and the corresponding resource use is detailed in Table 5. As shown in Table 5, resources were allocated efficiently, with the sample size set to 1000 to optimize resource utilization. The design is modular, allowing the sample size to be increased in increments of 1000 by reusing the IP cores for any sample size exceeding 1000. This modular approach enhances scalability and improves resource management.

Figure 14 presents a comparison of power consumption when executing the MCO algorithm on both the GPU and FPGA (proposed) platforms. The Y-axis represents power consumption in Watts (W). The predicted power use for the FPGA was measured using Vivado, while the GPU, an RTX3070, had its power use indirectly measured via NVIDIA's GeForce Experience tool [24]. The results indicate that the GPU consumed approximately 38W, while the FPGA consumed only 6.6 W. This shows that the GPU's power consumption was nearly six times higher than that of the FPGA.

4.5 Comparison Results with Parallelized PSO on the FPGA

The performance of MCO can be compared to that of Particle Swarm Optimization (PSO), as both are sampling-based optimization methods with similar characteristics [25]. Figure 15 illustrates the target-tracking results obtained using parallelized PSO on the FPGA, demonstrating that the PSO implementation was successful. When compared with the target-tracking results of the proposed MCO method shown in Fig. 11, the overall estimation outcomes were similar between the parallelized PSO and the proposed method. However, the convergence time for the parallelized PSO on the FPGA was faster than that of the proposed MCO method, which is an inherent difference between the two algorithms. Despite this difference, the proposed MCO method exhibited significantly shorter computation times than the parallelized PSO on the FPGA as shown in Fig. 16. The Y-axis represents computation time in milliseconds (ms).

5 Conclusions

This study introduces an FPGA-based parallelized MCO method designed for real-time tracking of targets along ballistic trajectories. By profiling and analyzing the computation times of the MCO method across various logical blocks, the proposed approach effectively accelerates the sampling and the likelihood calculation processes through the application of efficient pipelining techniques on an FPGA. Experimental results confirm that the proposed method significantly reduces computation time compared to the conventional MCO approach. Additionally, comparative analysis reveals that the proposed method offers superior performance, achieving faster computation times than a comparable FPGA-accelerated sampling-based optimization method. Furthermore, the proposed algorithm is well-suited for low-power guided munitions and other systems where active cooling is not feasible. Its efficient hardware acceleration minimizes power and thermal constraints, making it adaptable to high-speed, resource-limited applications such as autonomous aerial or space-based tracking systems.

Acknowledgements This work was supported in part by Theatre Defense Research Center funded by Defense Acquisition Program Administration under Grant UD240002SD, and in part by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2024-RS-2024-00438430) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation)."

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Bilik I, Tabrikian J (2010) Maneuvering target tracking in the presence of glint using the nonlinear Gaussian mixture Kalman filter. IEEE Trans Aerosp Electron Syst 46:246–262. https://doi.org/10.1109/taes.2010.5417160
2. Banani S, Masnadi-Shirazi M (2007) On tracking maneuvering targets in 3-dimensonal space with online observed colored glint noise parameter estimation. IEEE Int Conf Signal Process Commun 2007:1427–1430. https://doi.org/10.1109/icspc.2007.4728597
3. Yang H, Seong S (2022) Extended Kalman filter covariance tuning using LSTM for navigation system. J Inst Control Robot Syst 28(2):130–137. https://doi.org/10.5302/j.icros.2022.21.0192
4. Choi H, You S (2021) Design of GDR-based PF for electro-hydraulic nonlinear active suspension systems. J Inst Control Robot Syst 27(2):130–137. https://doi.org/10.5302/j.icros.2021.20.0135
5. Luo W, Sun P, Zhong F, Liu W, Zhang T, Wang Y (2020) End-to-end active object tracking and its real-world deployment via reinforcement learning. IEEE Trans Pattern Anal Mach Intell 42(6):1317–1332. https://doi.org/10.1109/TPAMI.2019.2899570
6. Bakirci M (2025) Vehicular mobility monitoring using remote sensing and deep learning on a UAV-based mobile computing platform. Measurement 244:116579. https://doi.org/10.1016/j.measurement.2024.116579
7. Park J, Lee M, Lee H, Hwang Y, Choi W, Jeong B (2023) Parallelized Monte carlo optimization with GPU for real-time ballistic target tracking. J Inst Control Robot Syst 29(8):607–619. https://doi.org/10.5302/j.icros.2023.23.0061
8. Ma Y, Suda N, Cao Y, Seo J, Vrudhula S (2016) Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), pp 1–8. https://doi.org/10.1109/fpl.2016.7577356
9. Sengupta J, Kubendran R, Neftci E, Andreou A (2020) High-speed, real-time, spike-based object tracking and path prediction

on google edge TPU. In: Proceedings of the 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp 134–135. https://doi.org/10.1109/AICAS48895.2020.9073867
10. Salamin S, Zervakis G, Klemme F, Kattan H, Chauhan Y, Henkel J, Amrouch H (2022) Impact of NCFET technology on eliminating the cooling cost and boosting the efficiency of google TPU. IEEE Trans Comput 71(4):906–918. https://doi.org/10.1109/TC.2021.3065454
11. Septier F, Cornebise J, Godsill S, Delignon Y (2011) A comparative study of Monte-Carlo methods for multitarget tracking. In: 2011 IEEE Statistical Signal Processing Workshop (SSP), pp 205–208. https://doi.org/10.1109/ssp.2011.5967660
12. Bruno M, Pavlov A (2005) Improved sequential Monte Carlo filtering for ballistic target tracking. IEEE Trans Aerosp Electron Syst 41(3):1103–1108. https://doi.org/10.1109/taes.2005.1541456
13. Zhou X, Lu Y, Lu J, Zhou J (2012) Abrupt motion tracking via intensively adaptive Markov-Chain Monte Carlo sampling. IEEE Trans Image Process 21(2):789–801. https://doi.org/10.1109/tip.2011.2168414
14. Zhang H, Nie G, Chen J, Zhang J, Yang G (2019) Uncertain motion tracking combined Markov Chain Monte Carlo and correlation filters. IEEE Access 7:167076–167088. https://doi.org/10.1109/access.2019.2953742
15. Khan Z, Balch T, Dellaert F (2005) MCMC-based particle filtering for tracking a variable number of interacting targets. IEEE Trans Pattern Anal Mach Intell 27(11):1805–1819. https://doi.org/10.1109/tpami.2005.223
16. Liu S, Mingas G, Bouganis C (2017) An unbiased MCMC FPGA-based accelerator in the land of custom precision arithmetic. IEEE Trans Comput 66(5):745–758. https://doi.org/10.1109/tc.2016.2630682
17. Tian X, Bouganis C (2011) A run-time adaptive FPGA architecture for Monte Carlo simulations. In: 2011 21st International Conference on Field Programmable Logic and Applications, pp 116–122. https://doi.org/10.1109/fpl.2011.30
18. Wang Y, Li P (2021) Algorithm and hardware co-design for FPGA acceleration of Hamiltonian Monte Carlo Based No-U-Turn Sampler. In: 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp 9–16. https://doi.org/10.1109/asap52443.2021.00009
19. Trzun Z, Vrdoljak M (2020) Monte Carlo simulation of missile trajectories dispersion due to imperfectly manufactured warhead. In: Proceedings of the 31st International DAAAM Symposium 2020, pp 0574–0583. https://doi.org/10.2507/31st.daaam.proceedings.079
20. Singer R (1970) Estimating optimal tracking filter performance for manned maneuvering targets. IEEE Trans Aerosp Electron Syst 4:473–483. https://doi.org/10.1109/taes.1970.310128
21. Li X, Jilkov V (2003) Survey of maneuvering target tracking. Part I. Dynamic models. IEEE Trans Aerosp Electron Syst 39(4):1333–1364. https://doi.org/10.1109/taes.2003.1261132
22. Kim J, Tandale M, Menon P (2010) Particle filter for ballistic target tracking with glint noise. J Guid Control Dyn 33(6):1918–1921. https://doi.org/10.2514/1.51000
23. Han Y, Lee H, Gwon H, Choi W, Jeong B (2022) Parallelized particle swarm optimization with GPU for real-time ballistic target tracking. IEMEK J Embed Syst Appl 17(6):355–365. https://doi.org/10.14372/IEMEK.2022.17.6.355
24. NVIDIA. "GeForce Experience." NVIDIA. https://www.nvidia.com/en-us/geforce/geforce-experience/. Accessed 3 June 2024
25. Park J, Lee H, Kwon H, Hwang Y, Choi W (2023) Parallelized particle swarm optimization on FPGA for realtime ballistic target tracking. Sensors 23(20):8456. https://doi.org/10.3390/s23208456