systems

MDPI

Article

DSDEVS-Based Simulation Acceleration with Event Filtering: USV Naval Combat Case

Juho Choi [1], Il-Chul Moon [2] and Jang Won Bae [3,*]

[1] Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea; 3747juho@gmail.com
[2] Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea
[3] School of Industrial Management, Korea University of Technology and Education (KOREATECH), 1600 Chungjeol-ro, Byeongcheon-myeon, Dongnam-gu, Cheonan-si 31253, Republic of Korea
* Correspondence: jangwon_bae@koreatech.ac.kr

Abstract

This study presents a DSDEVS-based method to accelerate simulation execution for AI training in USV (Unmanned Surface vehicle) naval combat scenarios. The proposed approach introduces an event filtering technique that selectively suppresses low-importance sensing events based on the distance to enemy targets. By dynamically adjusting structural couplings and modifying sensing frequency through domain-specific thresholds, the method reduces execution time while maintaining a balance between speed and fidelity. Two key parameters—Event Filtering Distance (EFD) and Sensor Acceleration Time Advance (SATA)—enable conditional event filtering and time advance adjustments within the sensor model. Experimental results demonstrate a 3.03 improvement in runtime, highlighting the effectiveness of the method and the trade-off between simulation speedup and fidelity.

Keywords: discrete event system; DEVS; DSDEVS; simulation for AI; simulation execution acceleration; unmanned surface vehicle simulation

Academic Editors: Ludovico Minati and Zhou He

Received: 27 August 2025
Revised: 28 October 2025
Accepted: 30 October 2025
Published: 2 November 2025

Citation: Choi, J.; Moon, I.-C.; Bae, J.W. DSDEVS-Based Simulation Acceleration with Event Filtering: USV Naval Combat Case. *Systems* **2025**, *13*, 979. https://doi.org/10.3390/systems13110979

1. Introduction

The Discrete Event System Specification (DEVS) formalism [1] provides a modular and hierarchical framework for modeling and simulating discrete event systems (DESs). As an event-driven formalism, it enables efficient execution by handling changes only at discrete event points making it particularly suitable for applications such as manufacturing systems, communication networks, logistics, and military simulations. DEVS promotes model reusability and composability, which enhances the scalability and maintainability of large-scale simulation models.

However, classical DEVS assumes a static structure throughout the simulation runtime. This limitation makes it challenging to model systems that inherently require dynamic structural changes, such as the creation or deletion of components, reconfiguration of subsystems, or changes in coupling relationships. As modern systems increasingly demand adaptive and reconfigurable behavior, this rigidity in structural modeling becomes a significant constraint on the applicability of conventional DEVS.

To overcome the rigidity of classical DEVS in dynamic environments, the Dynamic Structure Discrete Event System Specification (DSDEVS) [2] extends the DEVS formalism to support runtime structural reconfiguration. DSDEVS enables the dynamic addition or removal of model components and the changes of couplings, offering enhanced flexibility

and adaptability. These features make it particularly suitable for modeling autonomous systems, intelligent manufacturing, and adaptive traffic control. Moreover, DSDEVS has emerged as a promising simulation framework for reinforcement learning (RL) applications, where adaptability and efficient data generation are critical.

As simulation is increasingly used as a data generation tool for AI training, execution efficiency has become a critical concern. Training AI agents, particularly in complex multi-agent environments like naval combat, requires a large number of simulation episodes to ensure sufficient exploration and convergence. However, the computational cost of running these simulations—especially when using detailed models with real-time interactions—can become a major bottleneck in the overall training process. This necessitates simulation acceleration methods that can reduce execution time without compromising fidelity, enabling faster iteration and more scalable learning frameworks.

While DSDEVS provides powerful mechanisms for structural adaptation, simulations based on this formalism often suffer from increased modeling complexity and frequent inter-model communication, which can lead to long execution times. While this poses a performance challenge, we exploit the structural flexibility of DSDEVS to address it directly. We introduce a selective event sampling mechanism, referred to as Event Filtering, which dynamically adjusts couplings and message propagation during simulation to reduce computational overhead while preserving essential system behavior.

In particular, when DSDEVS-based simulations are used for reinforcement learning, the proposed event filtering directly enhances sample efficiency by reducing the wall-clock time of each training episode. In reinforcement learning (RL) environments, the efficiency of data generation is a key factor that determines the overall learning speed. Since proposed method, event filtering can reduce unnecessary event exchanges without altering the underlying model logic, they allow faster generation of training data through accelerated simulation runs. This enables agents to collect a larger number of interaction samples within the same computational budget, thereby improving sample efficiency in AI training. Therefore, event filtering can provide sample efficiency for faster training data generation and may be more effective in online reinforcement learning than offline reinforcement learning.

This research focuses on a distributed naval combat simulation scenario, where multiple friendly unmanned surface vehicles (USVs) are deployed to detect, track, and engage enemy USVs approaching a protected zone. Each USV operates autonomously based on local sensing and decision-making, with constraints on communication and sensor range.

This scenario captures key challenges of autonomous naval combat scenario, including decentralized control, limited sensing, real-time coordination, and perceptual uncertainty. These factors lead to frequent and dense message transmissions between sensor and control entities, which significantly increase computational demands during simulation. To address this issue, we propose an adaptive message filtering and structural control mechanism based on the DSDEVS formalism, termed Event Filtering, which selectively reduces simulation workload while preserving critical engagement dynamics.

In this study, we introduce a novel method for execution acceleration in DSDEVS-based simulations by leveraging event importance. Traditional DEVS simulations treat all events equally, resulting in high communication overhead. However, not every event significantly influences the simulation outcome. By prioritizing critical events and reducing the frequency of less impactful ones, our approach significantly reduces computational overhead while maintaining a high level of simulation fidelity. This enables more scalable and efficient simulation environments, especially for AI learning-based applications.

2. Backgrounds

2.1. DEVS and DSDEVS Formalism

Discrete Event System Specification (DEVS) is a formal framework designed to model discrete event systems (DES) with hierarchical and modular structures [1,3]. It defines system behavior using two core abstractions: the atomic model and the coupled model. By combining these two components, DEVS enables structured modeling of complex systems in a scalable and organized manner. The atomic model specifies the behavior of basic components, while the coupled model represents the composition and interaction of multiple atomic or coupled submodels. The standard specifications of these models are described as follows.

The formal definition of an atomic model (*AM*) in the DEVS formalism is given as:

$$AM = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$$

where

- X: Set of input events;
- Y: Set of output events;
- S: Set of states;
- $\delta_{ext} : Q \times X \to S$: External transition function, with $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$;
- $\delta_{int} : S \to S$: Internal transition function;
- $\lambda : S \to Y$: Output function;
- $ta : S \to \mathbb{R}_0^+ \cup \{\infty\}$: Time advance function.

The coupled model (*CM*), which organizes multiple atomic or coupled submodels into a single system, is formally defined as:

$$CM = \langle X, Y, M, EIC, EOC, IC, SELECT \rangle$$

where

- X: Set of external input events;
- Y: Set of output events;
- M: Set of component models;
- $EIC \subseteq X \times \bigcup_{m \in M} m.X$: External input coupling relations;
- $EOC \subseteq \bigcup_{m \in M} m.Y \times Y$: External output coupling relations;
- $IC \subseteq \bigcup_{m \in M} m.Y \times \bigcup_{n \in M} n.X$: Internal coupling relations;
- $SELECT : 2^M - \emptyset \to M$: Tie-breaking function for simultaneous events.

The Dynamic Structure DEVS (DSDEVS) formalism extends the classical DEVS framework to support structural change during simulation execution [2]. This allows models to dynamically add, remove, or reconfigure components and coupling relations within a DEVS coupled model. To enable this capability, DSDEVS introduces a new abstract entity called the *network executive*, which maintains the current structural configuration of the model network as part of its internal state.

A DSDEVS dynamic structure network, referred to as DSDEVN, is defined as follows:

$$DSDEVN = \langle \chi, M_\chi \rangle$$

where

- χ: The network executive;
- $M_\chi = \langle X_\chi, Y_\chi, S_\chi, \delta_{ext_\chi}, \delta_{int_\chi}, \lambda_\chi, ta_\chi \rangle$: The behavioral model of the executive χ;
- $S_\chi = \langle X_\Delta, Y_\Delta, M_\Delta, EIC_\Delta, EOC_\Delta, IC_\Delta, Select_\Delta \rangle$: The structural state of the DSDEVS model network.

2.2. Related Works

DEVS (Discrete Event System Specification) provides a modular and hierarchical framework for system modeling, and has been widely used across various simulation domains [1]. However, as the complexity of DES-based models increases, the corresponding growth in execution time significantly limits their applicability to large-scale or real-time scenarios, making execution time a key bottleneck. To address this challenge, numerous approaches have been proposed to accelerate DES-based simulations.

Distributed execution environments such as DEVS/CORBA were introduced to support scalable supply chain simulations [4], and Amdahl's law was revisited to analyze the theoretical performance limit of distributed simulation [5]. Several approaches have focused on leveraging parallel computing structures to accelerate simulation. Trabes and Wainer proposed a shared-memory parallel algorithm for DEVS simulations [6], while Liu and Wainer explored multi-grained parallelism in compute-intensive DEVS models [7]. Himmelspach et al. implemented a distributed architecture for Parallel DEVS models [8], and Cardenas et al. further enhanced performance using shared-memory-based DEVS execution algorithms [9].

Multi-resolution modeling (MRM) has also been introduced as a strategy to balance simulation accuracy and execution speed, by adjusting the level of model granularity [10,11]. These studies mainly focus on structural or platform-level enhancements to reduce the computational burden of large-scale DEVS models.

While the above studies focus on accelerating DEVS-based simulators, other approaches beyond the DEVS formalism have also been explored in the discrete-event simulation (DES) domain. For example, Wang and Tropper applied reinforcement learning to optimize time warp simulation [12], and NetSquid presented an efficient discrete-event simulation framework for quantum networks, demonstrating scalable and accurate event scheduling [13]. The DONS simulator was also introduced as a high-performance DES system using event coalescing and batching strategies [14]. While these approaches Ware not DEVS-specific, their principles inform the design of runtime event filtering and event suppression.

Surrogate-based optimization (SBO) has emerged as another class of techniques for improving simulation efficiency. Hong and Zhang [15] review the use of surrogate models to approximate expensive simulation outputs and guide optimization. Surrogate models have also been combined with evolutionary algorithms for high-dimensional, computation-intensive problems [16]. These methods reduce the number of expensive simulation evaluations by learning predictive models that estimate performance metrics.

More recent research explores the integration of DEVS and machine learning. Saadawi and Wainer [17] proposed a machine-learning-based regression model to approximate repeated DEVS components, thereby reducing computation, but did not engage in runtime event importance evaluation or suppression. Capocchi and Santucci [18] demonstrated the integration of DEVS-based simulation with reinforcement learning agents, showing that adaptive control can be embedded within DES environments.

While these prior studies have significantly advanced the performance and scalability of simulation systems, most of them focus on hardware-oriented parallelization strategies, static model abstraction, or simulation-level optimization. However, relatively little attention has been given to runtime event-level control based on contextual event importance—especially within scenarios requiring adaptive fidelity management such as real-time naval combat simulations. Furthermore, few approaches integrate dynamic structural adjustments at runtime based on learned or rule-based evaluation of message relevance.

To address these limitations, we propose an event filtering method (formerly referred to as event filtering) that suppresses low-importance events based on rule-based criteria. Unlike conventional static optimization techniques, this method leverages the capabilities

of DSDEVS to dynamically control couplings and filter events during runtime, enabling a more adaptive trade-off between computational efficiency and simulation fidelity.

In distributed simulation, HLA Data Distribution Management (DDM) provides a mechanism for controlling inter-federate data exchange by defining routing spaces and subscription regions. According to the IEEE 1516.2 Standard [19], DDM configurations such as attributes and routing spaces are specified in the Federation Object Model (FOM) prior to execution and remain fixed during runtime, permitting only the modification of region boundaries. As a result, DDM improves network-level communication efficiency but cannot modify the internal structure of simulation models.

Several studies have attempted to enhance the filtering efficiency of HLA DDM. Tan et al. [20] proposed an agent-based DDM that employs intelligent agents to filter data at the publisher side, thereby reducing redundant inter-federate messages. Their results demonstrated a significant reduction in communicated messages compared with traditional region-based and grid-based DDM, but the overall simulation processing time increased due to agent-management overhead. This finding indicates that DDM-based approaches are effective for improving communication efficiency yet limited in terms of runtime performance.

By contrast, the proposed DSDEVS-based event-filtering method operates within a single simulator, where coupling relationships between models can be changed during simulation execution to control event propagation. This model-level structural control reduces unnecessary event generation before propagation occurs, directly improving computational efficiency and simulation runtime. Such execution efficiency is particularly advantageous in AI-training environments, where simulations must be executed repeatedly for policy learning and validation.

This study addresses the USV combat system. Recent studies have focused on improving the autonomy and coordination of unmanned maritime systems through advanced control strategies. For example, prescribed-performance and robust control schemes have been applied to USV and USV–UAV cooperation problems to guarantee bounded tracking errors and stable coordination under environmental disturbances [21–23]. These works highlight ongoing progress in ensuring stability and performance for autonomous vehicle operations in maritime environments. While such control-oriented research focuses on agent-level behavior design, the present study complements this direction by developing a scalable simulation framework that enables rapid experimentation and validation of these algorithms.

3. USV Naval Combat Model Development

In this study, we developed a DES based naval combat simulator with DEVS formalism to serve as a testbed for execution efficiency optimization. The simulator models combat scenarios involving friendly Unmanned Surface Vehicles (USVs) and enemy USVs. The core purpose of the simulation is to capture the operational dynamics of USVs as they detect, maneuver toward, and engage enemy USVs using onboard sensors and autonomous decision-making. The system is designed to support large-scale engagements involving numerous agents, enabling the study of distributed behavior in naval warfare contexts. The ultimate objective of the simulator is to support the development of optimal engagement strategies by training AI-based algorithms within the simulated environment. This goal necessitates a large number of simulation runs, making execution efficiency a critical requirement for the system.

3.1. USV Naval Combat Scenario

The simulation scenario employed in this study represents a distributed naval engagement environment. In this setup, 20 friendly unmanned surface vehicles (USVs) are deployed with the objective of intercepting 20 approaching enemy USVs before they reach designated target areas. Each USV autonomously detects, tracks, and engages enemy units using onboard sensors and control logic. Figure 1 shows the progress of the scenario.

At the beginning of the simulation, the enemy USVs depart from predefined locations and move toward their respective destinations. These adversarial units do not perform detection or counterattacks; rather, they follow static, predetermined trajectories. In Figure 1, the enemy USVs can be seen maneuvering along orange trajectories from their initial positions (red circles) toward their target positions (gray squares).

Meanwhile, the 20 friendly USVs are positioned across the sea and begin to operate as soon as the simulation starts, using their sensors to detect and pursue enemy USVs to eliminate the threat. In Figure 1, 20 friendly USVs, indicated by blue dots, are shown waiting at their initial positions. The pink line denotes the operational boundary that should not be crossed prior to mission commencement; after the operation begins, this boundary is redefined as the permissible area.

Each friendly USV determines its engagement target through a distance-based selection process. For every adversarial USV detected within the sensor range, the Euclidean distance is calculated. Among the detected set, the adversarial unit with the minimum distance is designated as the target. The movement objective of the friendly USV is updated toward the position of this target, and firing is initiated once the designated adversarial unit enters the engagement range. The USVs are initialized with 200 rounds of ammunition and fire one round per second at their selected target. The probability of a successful hit is determined by predefined firing parameters, which consider factors such as the distance and relative orientation to the target. When an enemy USV is hit, it is considered neutralized and remains stationary at its last known position. The simulation concludes when all 20 enemy USVs are either neutralized or have successfully reached their respective target positions (gray squares in Figure 1).

Each USV selects its target independently based on proximity. Consequently, multiple friendly USVs may engage the same adversarial unit simultaneously, resulting in many-to-one engagements. This behavior arises naturally from the independent decision-making process of each USV. Although mission assignment or conflict resolution algorithms could be introduced in future work to improve the scenario, the contribution of this study lies in improving the execution of a fixed scenario rather than modifying the scenario itself.

This scenario encapsulates the essential characteristics of decentralized, multi-agent maritime operations, where each USV makes decisions based solely on local sensing data and event-driven control logic. The simulation is implemented using a fully event-driven DSDEVS model, in which communication between system components—such as sensor and maneuver modules—occurs via frequent event messages.

While this modeling approach enables high-fidelity representation of engagement dynamics, it also introduces considerable execution overhead, particularly as the number of agents increases.

Therefore, the main challenge lies not only in accurately modeling the combat behavior, but also in efficiently managing the simulation execution under high message density. This makes the scenario a suitable testbed for evaluating event importance estimation and coupling regulation techniques.

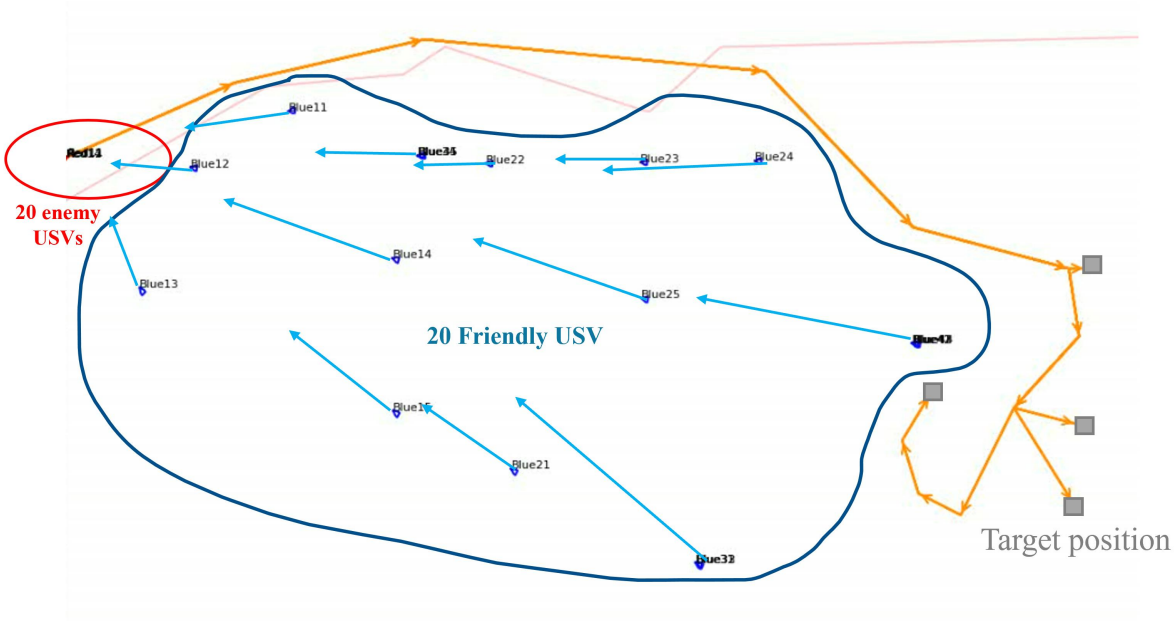


Figure 1. USV naval combat scenario: 20 enemy USVs follow the orange trajectory and 20 USVs follow enemy USVs to neutralize enemy USVs.

3.2. USV Naval Combat System Modeling

The simulation adopts a hierarchical modeling framework built upon the DSDEVS formalism to realistically represent the complexity of maritime combat environments. As an extension of the traditional DEVS methodology, DSDEVS enables structural reconfiguration during simulation runtime, providing greater flexibility and scalability in dynamic operational settings.

The simulation model constructed in this work is composed primarily of friendly USV entities and enemy USV entities, organized in a hierarchical manner to represent their operational interactions. This layered architecture facilitates modular development and clear separation of functional components, allowing for seamless integration of sensing, decision-making, and engagement behaviors within the simulation environment.

3.2.1. Structural Modeling

The hierarchical modeling structure as depicted in Figure 2, supports multi-level abstraction, capturing both the behavior of individual units (USVs and enemy USVs) and the overall dynamics of the simulated battlespace. At the top-level root model, global environmental parameters and the general combat scenario are defined. The middle layer governs the strategic navigation and mission-level behavior of each platform. At the lowest level, fine-grained behaviors are implemented using atomic models, providing detailed control over individual entity actions.

Each USV and enemy USV in the simulation shares a common structural model, but they differ in the internal logic of their Command & Control (C2) atomic models. The USV model incorporates a more sophisticated decision-making system that allows it to detect enemy movements and respond accordingly. In contrast, the enemy USV model follows a predetermined path toward its objective, operating with a simplified control mechanism.

As illustrated in Figure 3, these entities interact through the exchange of Maneuver-State and DamageState messages, enabling the simulation to reflect real-time engagement dynamics with high accuracy.

In the proposed model, each USV updates and transmits its position information at every internal transition, which occurs according to the model's predefined time advance. At each such transition, a maneuver state event containing the current position of the

USV is generated and broadcast to all adversarial units. This broadcasting is applied uniformly to all adversarial units, and the transmission interval remains constant throughout the simulation.

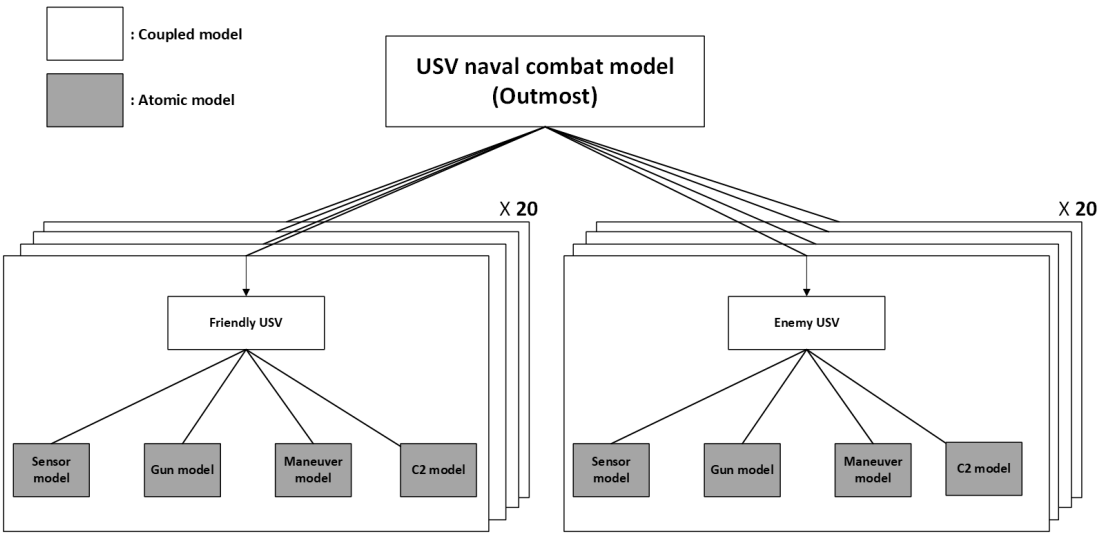


Figure 2. Hierarchical structure of USV naval combat model.

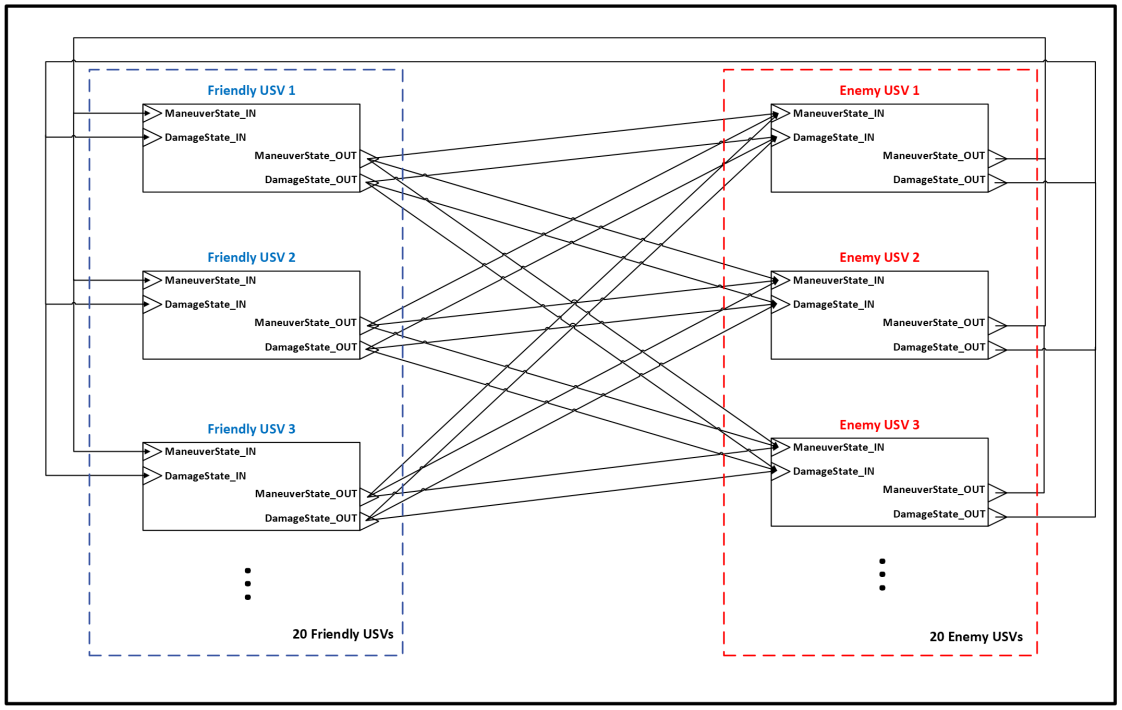


Figure 3. DEVS coupled Diagram for USV naval combat outmost model. For clarity, only representative USVs are illustrated; the actual model includes 20 friendly and 20 enemy USVs. The blue arrows indicate the maneuvers of the friendly USVs, and the blue area represents their initial standby region defined by the scenario.

3.2.2. Behavior Modeling

As illustrated in Figure 4, the USV coupled model consists of four atomic models—*Sensor*, *Gun*, *Maneuver*, and *C2* (Command & Control)—each responsible for a specific behavioral function. These atomic models represent the leaf nodes in the hierarchical structure (Figure 2) and operate via event-driven state transitions. The couplings among

them collectively define the behavior of an individual USV, and the overall simulation progresses through the interaction of these components.

The Sensor model, shown in Figure 5, begins in the Wait state. Upon receiving a ManeuverState input event, it performs an external transition to the Sensing others state and stores the received data. The model also receives its own motion information through the MyManeuverState input port. Using this data, the model computes the distances to enemy USVs and determines whether they are within radar detection range, which reflects real-world sensing capabilities. Based on this assessment, it performs an internal transition and outputs the sensing result via the ManeuverState_OUT port as a MsgSensorState message, which contains the observed maneuver states of enemy units.

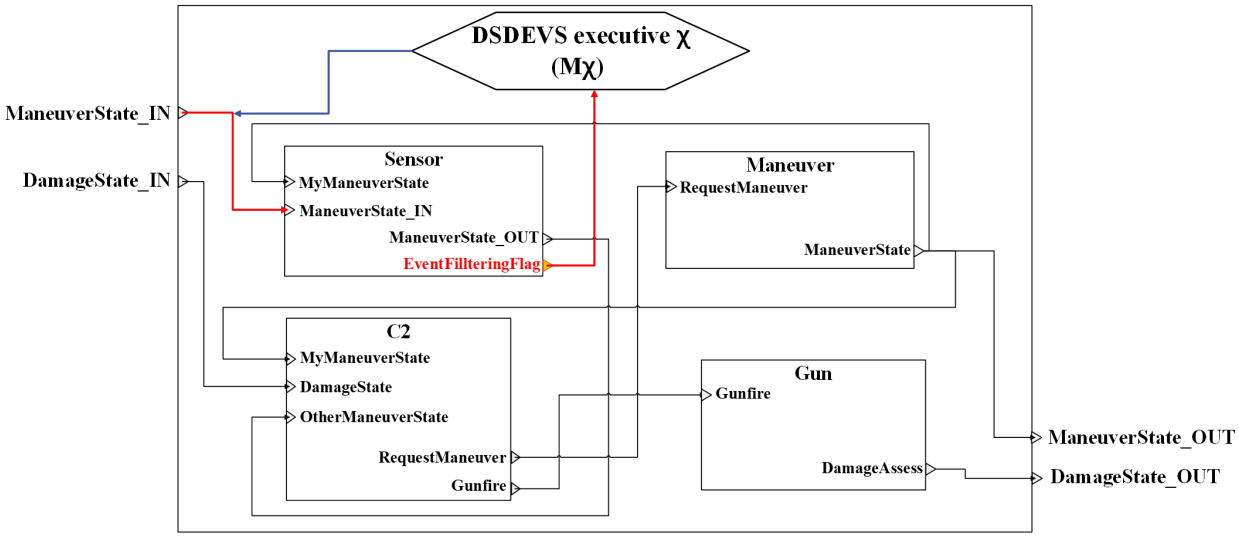


Figure 4. Diagram of USV Coupled model. The red line depicts couplings affected by the DSDEVS executive.The executive that receives the message from EventFilteringFlag controls the sensor message coupling according to its content (blue line).

The Maneuver model receives target maneuver commands from the C2 model and updates the USV's state accordingly. The RequestManeuver input port accepts a MsgReq message containing maneuver instructions. Based on this message, the model calculates and updates the next position and velocity of the USV. The updated state is sent through the ManeuverState output port as a MsgManeuverState, which is transmitted to the Sensor and C2 models as well as the external output port. The internal structure of the Maneuver model is illustrated in Figure A1. The dynamic behavior of the USV is modeled using a simplified kinematic formulation driven by thrust and bank commands. At each simulation step, the position is updated from the current velocity,

$$x_{t+1} = x_t + v_x \Delta t, \quad y_{t+1} = y_t + v_y \Delta t,$$

where (x, y) denote the position and (v_x, v_y) the velocity components. The forward speed is updated as

$$v_{t+1} = v \cdot f + T \Delta t,$$

where v is the current speed, f is the friction coefficient, and T is the thrust command. The velocity components are then determined from the updated speed and bank command,

$$v_x = v_{t+1} \cos(\psi + \phi), \quad v_y = v_{t+1} \sin(\psi + \phi),$$

where ψ is the yaw angle and ϕ is the bank command. The yaw is recalculated from the velocity vector as

$$\psi = \arctan\left(\frac{v_y}{v_x}\right).$$

Through this formulation, the USV steers toward the commanded maneuver direction while accounting for thrust and frictional effects. This representation balances physical realism with computational efficiency, making it suitable for large-scale naval combat simulations.

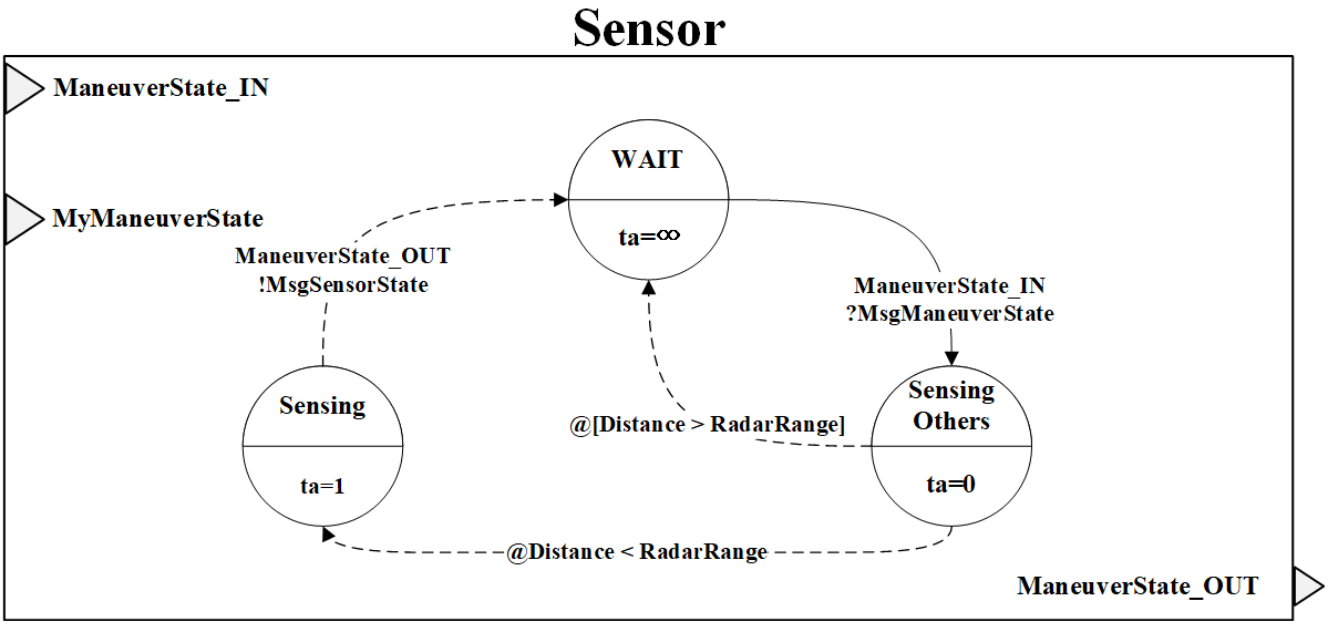


Figure 5. Diagram of Sensor Atomic model.

The Gun model initiates its behavior when a firing command message (MsgGunFire) is received through the Gunfire port. The `Decide_Fire()` function evaluates whether to fire based on the command contents, triggering an external transition to the Fire state. After a predefined delay in the Fire state, the model returns to the Wait state via an internal transition. If the firing condition is satisfied, a MsgDamageAssess message is generated and sent to the corresponding enemy SV through the DamageAssess port. The internal structure of the Gun model is provided in Figure A2.

The C2 (Command and Control) model synthesizes situational awareness by combining enemy position data from the OtherManeuverState port and self-state data from the MyManeuverState port. This information is processed by the `produce_behavior` function, which generates maneuver and firing commands. Target maneuver commands are transmitted to the Maneuver model, while firing commands are sent to the Gun model. It also receives damage reports from the DamageState port and handles unit neutralization if necessary. A diagram of the C2 model is shown in Figure A3. In contrast, the enemy USV model uses a simplified C2 mechanism. It does not rely on any perception input; instead, it generates maneuver commands based solely on predefined trajectories and does not issue firing commands.

3.3. Event Filtering for Simulation Acceleration

In the proposed simulation, all USVs detect the positions of enemy USVs at fixed intervals, continuously updating sensing data. While this ensures responsiveness, it also increases message traffic and computational load, which may significantly degrade simulation performance—especially in scenarios requiring real-time operation or multiple simulation runs for AI training. In particular, detecting distant enemy USVs with a low

probability of engagement may introduce unnecessary computational burden. To address this issue, this study applies the Event Filtering method, which dynamically adjusts the sensing frequency based on the relative importance of enemy targets. To address this issue, we apply an event filtering method, which dynamically regulates sensing frequency based on the relative importance of enemy targets. Specifically, event filtering refers to a mechanism that enables or disables couplings based on internal model states or scenario-specific conditions.

Figure 6 illustrates how DSDEVS enables dynamic structural modifications through the event filtering process. The operation can be understood as a sequence of steps ①–⑥ depicted in the figure. First, when a message arrives at the ManeuverState_IN port of the Sensor model ①, its content is examined, and the EventFilteringFlag port sends a triggering message to the DSDEVS executive ②. The executive then removes the coupling from ManeuverState_IN to Sensor.ManeuverState_IN ③, suppressing unnecessary message traffic.

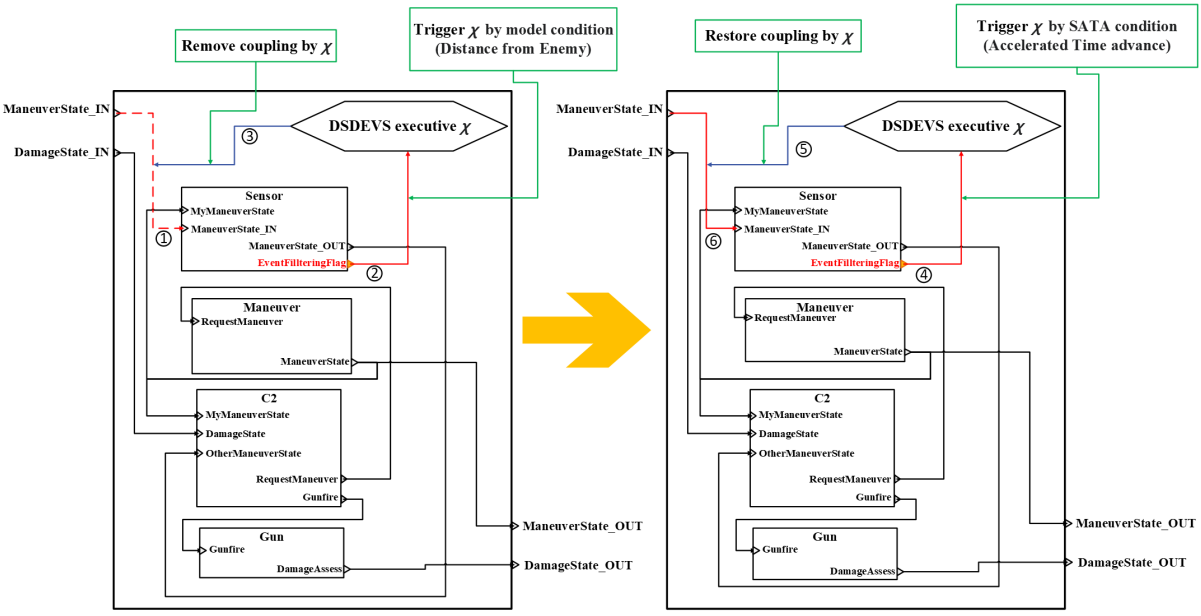


Figure 6. Dynamic structural modification through event filtering. The operation can be understood as a sequence of steps ①–⑥ depicted in the figure. The red lines highlight the couplings involved in each step, while the dashed lines indicate the couplings that have been removed. (**Left**) Event filtering ON: messages arriving at the ManeuverState_IN port trigger the Sensor.EventFilteringFlag, which signals the DSDEVS executive to remove the coupling and suppress message transmission. (**Right**) Event filtering OFF: after the time advance condition elapses, the EventFilteringFlag triggers the executive to restore the coupling, allowing messages to flow again.

After the predefined time advance elapses, the EventFilteringFlag again signals the executive ④, which restores the previously removed coupling ⑤. Messages can then resume flowing into the Sensor model through the reconnected ManeuverState_IN port ⑥.

This sequence of steps represents how event filtering is applied during simulation: the Sensor model monitors incoming messages and signals the executive, while the executive dynamically removes and restores couplings to regulate event propagation without altering the overall model definition.

The Event Filtering method modifies the sensing interval based on the distance between a USV and an enemy USV while dynamically regulating message transmission when necessary. To implement this, the conventional static sensor model was transformed into a DSDEVS-based model. By leveraging DSDEVS, the sensing mechanism and structure of

the sensor model can be flexibly adjusted during simulation execution, thereby reducing unnecessary computations while minimizing the impact on simulation accuracy.

A specific threshold, referred to as the Event Filtering Distance (EFD), is defined, and the sensing frequency is adjusted accordingly. USVs normally detect enemy USVs at a standard rate, but when an enemy USV is located beyond the EFD, the sensing frequency is reduced, or message transmission is temporarily suspended. To reduce the sensing frequency, the time advance of the sensor model is increased during this process. This increased time advance is referred to as SATA (Sensing Acceleration Time Advance).Conversely, when an enemy USV enters the EFD, the detection interval returns to its normal state.

During the initial phase, when enemy USVs are still approaching from a long distance, most of them remain outside the EFD, leading to a reduction in detection events and message transmissions. However, as enemy USVs enter the engagement range, detection accuracy becomes critical, and the normal sensing cycle is applied to those within the EFD.

In the process of applying the event filtering method, DSDEVS-based structural change are performed. When an enemy USV is located outside the EFD, the simulator disables the coupling between USVmodel.ManeuverState_IN and Sensor.ManeuverState_IN, effectively suspending message transmission for the duration defined by SATA. This dynamically increases the sensing interval. Conversely, when the enemy enters the EFD, normal sensing resumes, and the coupling is reactivated.

The event filtering method aims to reduce simulation calculation load and improve execution speed by selectively suppressing low-importance events. In this simulator, EFD serves as the threshold for applying event filtering, determined based on the domain characteristics and modeling objectives. Since the simulator is designed to simulate the autonomous decision-making of friendly USVs, EFD is used to identify events unlikely to influence simulation outcomes. SATA complements this by adjusting the time advance of atomic models, reducing internal transitions triggered by those low-importance events.

The dynamic coupling process between the sensor atomic model and the DSDEVS executive is summarized in Algorithm 1. The algorithm operates in a unified event-driven loop where both external and internal events are sequentially processed. When the distance between a sensor S_i and an adversarial unit exceeds the predefined Event Filtering Distance (EFD), the sensor triggers an `EventFilteringFlag` and holds its operation for the duration of SATA. Upon receiving this signal, the executive temporarily removes all inbound couplings to S_i.`ManeuverState_IN`, thereby suspending unnecessary event propagation. Once the holding period expires, the couplings are restored, ensuring that the overall system behavior remains consistent while reducing computational overhead during inactive sensing periods.

Algorithm 1: Single-Loop Dynamic Event Filtering and Coupling Control for Sensor S_i.

Input: Event list (internal/external), simulation end time $T_{\max}$
Data: Event Filtering Distance (EFD), Sensor Acceleration Time Advance (SATA)
$\texttt{flag}_{S_i} \in \{\texttt{TRUE}, \texttt{FALSE}\} \leftarrow \texttt{FALSE}$;
temporary removal list $\mathcal{R}(S_i) \leftarrow \emptyset$;
nominal time advance $\tau_{S_i}^{\text{nom}}$;
current time $t \leftarrow 0$
while $t < T_{\max}$ **do**
  (1) Retrieve the next event:
  Extract the earliest scheduled event (e, t_k) from the event list and set $t \leftarrow t_k$.
  (2) Process external input (message arrival):
  if e is an external event targeting S_i.*ManeuverState_IN* **then**
    Compute the distance $d_{ij}(t)$ between S_i and the message source.
    if $d_{ij}(t) > EFD$ **then**
      Set $\texttt{flag}_{S_i} \leftarrow \texttt{TRUE}$.
      Update the sensor time advance to $\tau_{S_i} \leftarrow$ SATA (temporary hold).
      Skip message propagation during this period and retain only minimal internal updates.
    end
    else
      Process the message normally without filtering or structural change.
    end
  end
  (3) Process internal transition (timer expiration):
  if *e is an* internal *event and its target model is* S_i **then**
    if $\tau_{S_i} =$ SATA **then**
      Set $\texttt{flag}_{S_i} \leftarrow \texttt{FALSE}$ and restore $\tau_{S_i} \leftarrow \tau_{S_i}^{\text{nom}}$.
    end
    Update the internal state of S_i.
  end
  (4) Executive-level structural control:
  if $\texttt{flag}_{S_i}$ *changed to* `TRUE` **then**
    Identify all inbound coupling edges $\mathcal{E}_{\text{in}}(S_i)$ connected to S_i.`ManeuverState_IN`.
    Remove each edge from the active coupling set and record it in $\mathcal{R}(S_i)$.
  end
  else if $\texttt{flag}_{S_i}$ *changed to* `FALSE` **then**
    Restore all coupling edges stored in $\mathcal{R}(S_i)$.
    Clear $\mathcal{R}(S_i)$ after restoration.
  end
  (5) Advance time: Move to the next earliest event and repeat.
end

4. Case Study

In this chapter, we conduct a case study using the simulator defined in the previous section. Specifically, we investigate how the two features—EFD (Event Filtering Distance) and SATA (Sensing Acceleration Time Advance)—affect simulation outcomes and execution time when applying Event Filtering to the scenario. The objective is to analyze their

influence and provide guidance on selecting appropriate EFD and SATA values to support decision-making within the given scenario.

4.1. Regression Analysis of Event Filtering

This section presents a case study designed to evaluate acceleration performance and fidelity degradation based on acceleration parameters. Specifically, we examine the impact of the Event Filtering method by testing a total of 48 cases, consisting of 8 EFD (Event Filtering Distance) settings and 6 SATA (Sensing Acceleration Time Advance) settings, each replicated 40 times. Experimental results are recorded as CSV logs, and performance measures are derived through log analysis.

Table 1 presents the experimental design, including variable names, variation cases, and the implementation details of each variable.

Table 1. Experimental design of Case Study.

Variable Name (Unit)	Variation Cases	Implements
EFD (m)	1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000	Distance threshold of operation event filtering
SATA (-)	1, 20, 40, 60, 80, 100	The amount of accelerated time advance activated when event filtering is activated
Total cases	$8 \times 6 = 48$ cases	
Complete Experiment Design	48 cases $\times$ 40 replications = 1920 experiments	

Additionally, this study proposes two performance measures to evaluate simulation speedup and fidelity: Simulation Execution Time (runtime) and Mean Trajectory Distance Discrepancy (MTDD).

Simulation execution time (runtime) is used as the primary performance metric for the proposed Event Filtering method. This measure assesses the efficiency of the simulation execution optimization approach. Under identical hardware conditions, a reduction in simulation execution time indicates that the simulation was executed with a lower computational load.

Additionally, the impact of the optimization method on simulation results is evaluated as a measure of fidelity degradation. Specifically, this study examines the difference in the trajectories of 20 friendly USV models before and after applying the proposed method. This difference is quantified using the Mean Trajectory Distance Discrepancy (MTDD), which serves as an indicator of fidelity loss. The Dynamic Time Warping (DTW) algorithm [24] is employed to compute MTDD. DTW is selected because it can robustly compare two trajectories even when they differ in length or temporal alignment. In naval combat simulations, the movement patterns of USVs may exhibit time shifts or speed variations due to stochastic sensing and decision-making processes. A direct point-to-point comparison at identical time steps could misrepresent the similarity between trajectories when the same maneuver occurs at slightly different times. DTW addresses this by performing a non-linear alignment that minimizes cumulative spatial distance while allowing flexible temporal matching, making it particularly suitable for fidelity assessment in such scenarios.

Although outcome-based indicators such as hit or survival rate also exhibited observable variations under event filtering, their changes were not always proportional to the degree of structural modification. In several cases, different filtering configurations produced nearly identical engagement results despite distinct maneuver patterns, indicating that such discrete outcomes are less effective for analyzing incremental fidelity degradation. In this study, our focus was on examining micro-level behavioral fidelity—that is, how the trajectories of individual agents evolve when event filtering alters their sensing and maneuvering behavior. For this purpose, the Mean Trajectory Distance Discrepancy (MTDD) was selected, as it provides a continuous and sensitive measure that correlates

with the extent of message suppression and captures subtle kinematic deviations between baseline and filtered runs. Nevertheless, MTDD mainly reflects geometric and temporal consistency rather than overall mission success. A combined framework that integrates trajectory-based and outcome-based indicators will therefore be explored in future research to enable a more comprehensive multi-scale assessment of simulation fidelity.

The MTDD for a simulation case 'c', where an event filtering method is applied, is defined as follows:

$$P_{n,i} = \{(x_{n,i}(t), y_{n,i}(t)) \mid t = 0, 1, \ldots, T_n\} \tag{1}$$

$$P_{c,i} = \{(x_{c,i}(t), y_{c,i}(t)) \mid t = 0, 1, \ldots, T_c\} \tag{2}$$

$$d_c(i) = DTW(P_{n,i}, P_{c,i}) \tag{3}$$

$$MTDD_c = \frac{1}{B} \sum_{i \in Blues} d_c(i) \tag{4}$$

where

- $Blues$: A set of IDs of friendly USVs.
- i: Component of $Blues$, which is ID of USV.
- B: Number of $Blues$ components.
- n: Length of DTW matching path.
- c: Experiment case, n: Non-accelerated case.
- $x_{c,i}(t), y_{c,i}(t)$: Longitude, Latitude of USV whose ID is i, at time t and case c.

The data collected from the experiments were analyzed to evaluate the impact of acceleration parameters on simulation execution runtime and fidelity. A response surface was plotted in Figures 7 and 8 to examine the trends in simulation runtime with respect to SATA and EFD. The results indicate that simulation runtime decreases as SATA increases and EFD decreases.

However, when SATA and EFD exceed a certain range, runtime begins to increase. This suggests that excessive adjustments to acceleration parameters can have negative effects. From the experimental data, it was observed that SATA values between 20–60 (s) and EFD values between 2000–5000 (m) demonstrated notable acceleration efficiency.

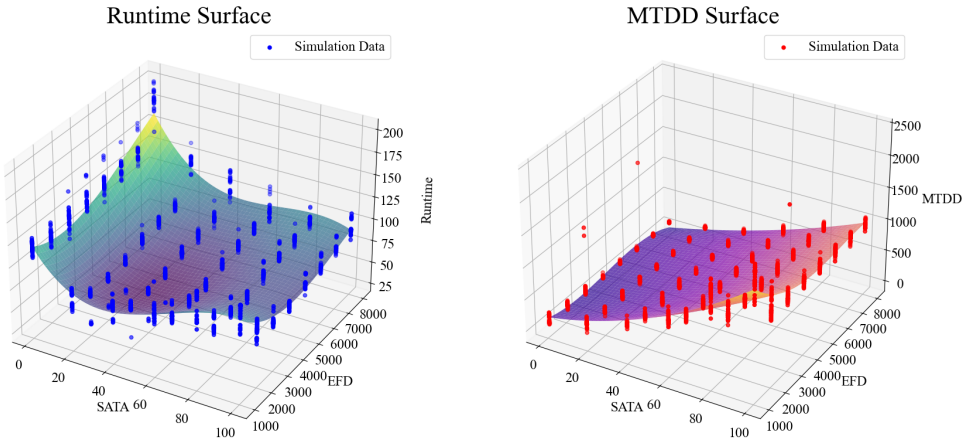


Figure 7. Response surfaces of case study result. Left: Response surface of Simulation Execution Runtime. Right: Response surface of Mean Trajectory Distance Discrepancy (MTDD).

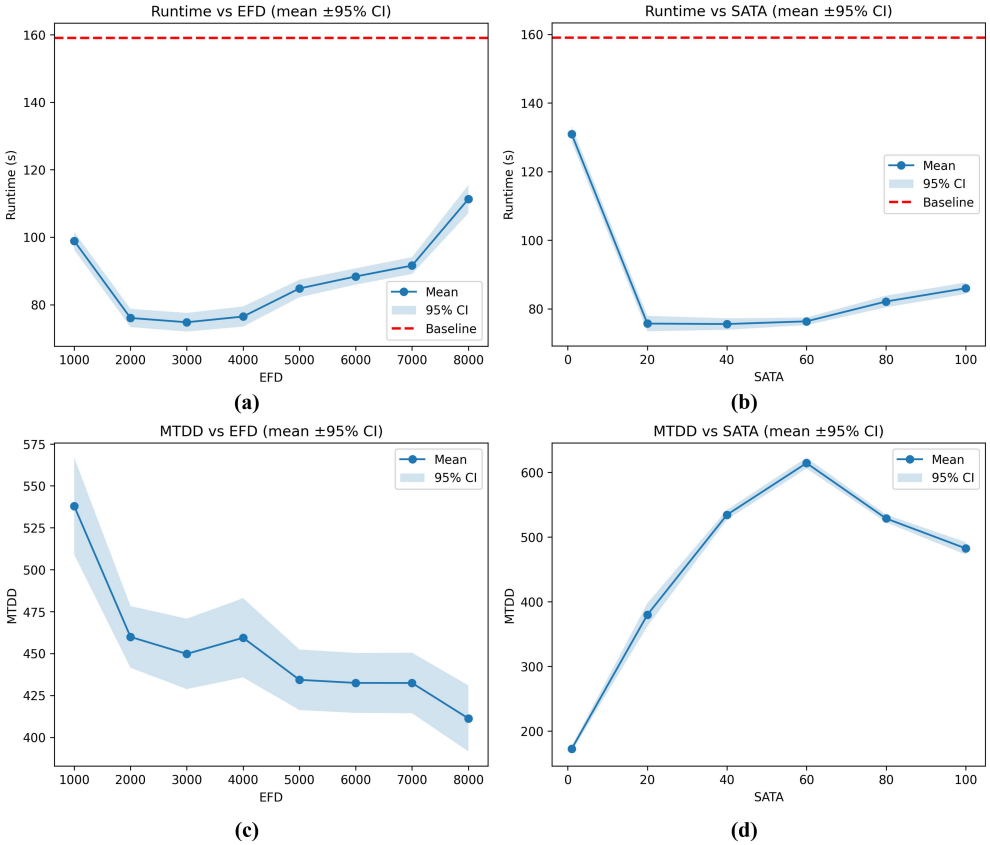


Figure 8. Runtime and MTDD results under different parameter settings. The mean values (solid lines) and 95% confidence intervals (shaded areas) are shown for each factor: (**a**) Runtime vs. EFD, (**b**) Runtime vs. SATA, (**c**) MTDD vs. EFD, and (**d**) MTDD vs. SATA. Red dashed lines indicate the baseline values (non-accelerated configuration).

A response surface for the fidelity measure MTDD is presented in Figure 7. The results clearly show that as SATA increases and EFD decreases, MTDD increases significantly, indicating a trade-off between acceleration efficiency and fidelity. As shown in Table 2, when a second-order polynomial regression was applied, it was observed that SATA has a dominant influence on the increase in MTDD.

Table 2. Result of Regression analysis for Runtime and MTDD.

Dependent Variable (Model)	Experimental Variable	Coef.	$p > \|t\|$	R^2 (Adj.R^2)
Runtime (linear)	SATA	−0.3825	0.000	0.198 (0.197)
	EFD	0.2264	0.000	
Runtime (2nd polynomial)	SATA	−0.3866	0.000	0.397 (0.396)
	EFD	0.2264	0.000	
	$SATA^2$	0.2475	0.000	
	SATA × EFD	−0.2683	0.000	
	EFD^2	0.0399	0.015	
MTDD (linear)	SATA	0.8921	0.000	0.831 (0.831)
	EFD	−0.1872	0.000	
MTDD (2nd polynomial)	SATA	0.8910	0.000	0.859 (0.859)
	EFD	−0.1872	0.000	
	$SATA^2$	0.0623	0.000	
	SATA × EFD	−0.1270	0.000	
	EFD^2	0.0301	0.000	

Table 2 presents the results of a regression analysis used to construct a meta-model that evaluates how well the acceleration parameters (EFD and SATA) explain the performance measures (runtime and MTDD). The regression coefficients for the runtime model confirm that runtime decreases as SATA increases and EFD decreases.

To examine the statistical significance of parameter effects, a two-way ANOVA was conducted with Event Filtering Distance (EFD) and Sensor Acceleration Time Advance (SATA) as factors. The results, summarized in Table 3, show that both main effects and their interaction significantly influenced simulation runtime and fidelity.

Table 3. Two-way ANOVA results showing the effects of Event Filtering Distance (EFD) and Sensor Acceleration Time Advance (SATA) on simulation runtime and fidelity (MTDD). All effects were statistically significant at $p < 0.001$, based on 40 replications per configuration.

Dependent Variable	Source	Sum of Squares	df	Mean Square	F	*p*-Value
Runtime	EFD	2.72×10^5	7	3.89×10^4	662.079	<0.001
	SATA	7.42×10^5	5	1.48×10^5	2528.837	<0.001
	EFD×SATA	1.45×10^5	35	4.13×10^3	70.393	<0.001
	Residual	1.10×10^5	1872	5.87×10^1		
MTDD	EFD	1.92×10^7	7	2.74×10^6	336.231	<0.001
	SATA	2.26×10^8	5	4.53×10^7	5552.476	<0.001
	EFD×SATA	1.40×10^7	35	4.01×10^5	49.163	<0.001
	Residual	1.53×10^7	1872	8.16×10^3		

Note: Two-way ANOVA with 40 replications per configuration. *p*-values < 0.001 are shown as "<0.001".

For runtime, both EFD and SATA exhibited strong effects ($F = 662.079$ and 2528.837, respectively, $p < 0.001$), while their interaction was also significant ($F = 70.393$, $p < 0.001$).

Similarly, for the fidelity metric (MTDD), both EFD and SATA had statistically significant impacts ($F = 336.231$ and 5552.476, $p < 0.001$), along with their interaction term ($F = 49.163$, $p < 0.001$).

These results confirm that variations in filtering distance and time advance parameters meaningfully affect both the execution performance and trajectory-level fidelity of the simulation. Overall, the ANOVA results quantitatively validate that the proposed event-filtering mechanism is sensitive to the parameter configuration, supporting the experimental trends observed in Figures 7 and 8.

4.2. Detailed Analysis for Case Study

To further investigate the relationship between the previously analyzed acceleration parameters and performance measures, we categorize the acceleration parameters into three levels and examine their effects. Table 4 presents the EFD and SATA values extracted from the experimental design, classified into three acceleration levels: weak, normal, and strong.

Table 4. Cases extracted based on the level of influence of parameters in the experimental results.

Level	EFD	SATA
Weak	8000	1
Normal	5000	40
Strong	1000	80
EFD 3 case × SATA 3 case = 9 case		

A scatter plot illustrating runtime and MTDD for the nine extracted cases is shown in Figure 9, with each case distinguished by color. Figure 10 presents density histograms for runtime and MTDD across these cases. In the 'EFD weak-SATA weak' case (red),

MTDD remains around 100, but the runtime is relatively high. As the EFD acceleration level increases (i.e., EFD decreases) through 'EFD normal-SATA weak' (orange) and 'EFD strong-SATA weak' (pink), runtime improves by an average of 1.41 and 1.49 times, respectively. Conversely, as the SATA acceleration level increases (i.e., SATA increases) through 'EFD weak-SATA normal' (blue) and 'EFD weak-SATA strong' (green), runtime improves significantly by an average of 1.70 and 2.02 times. However, this improvement comes at the cost of fidelity, as MTDD increases substantially by an average of 3.31 and 8.54 times, respectively, indicating a notable fidelity loss.

Figure 11 depicts the simulation state at the moment when the trajectory error increases in the latter half of the simulation, transitioning from 'EFD normal-SATA strong' to 'EFD strong-SATA strong' (as shown in Figure 9). When both SATA and EFD acceleration levels are excessively high, it has a significant impact on sensing, which in turn affects both the tracking performance and firing performance of the USV model. As a result, enemy USVs fail to be neutralized, leading to substantial trajectory discrepancies compared to non-accelerated cases.

At higher EFD settings, some enemy USVs were not neutralized during the final stage of engagement. This is because as the filtering interval gets longer, the detection update frequency decreases too much, causing friendly USVs to lose target information when the target leaves the effective detection range. As a result, certain enemy USVs remained undetected or were re-acquired too late for successful interception. This behavioral outcome highlights the trade-off between acceleration and situational awareness in event-filtered simulations.

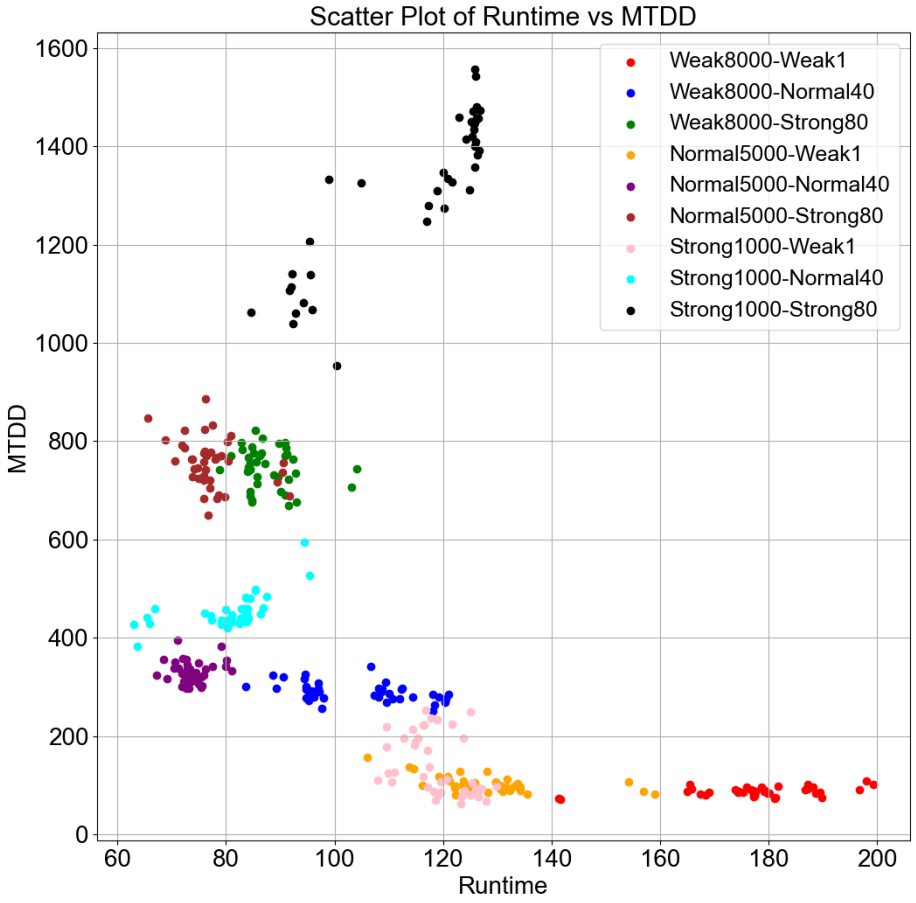


Figure 9. Scatter plot of Runtime and MTDD.

Figure 12 illustrates the trajectory error over time for each acceleration level. The trajectory error is computed by dividing the simulation time into 100-s intervals and calculating MTDD for each segment. The results show that an increase in SATA has a significant impact on trajectory error, which aligns with the analysis in Table 4, where the coefficient for SATA is much larger than that for EFD. In the 'EFD strong-SATA strong' case, trajectory error in the later stages of the simulation is noticeably higher compared to cases where EFD is at normal or weak levels. Notably, around the 2000-step mark, situations arise where a small EFD value significantly affects simulation fidelity. To further investigate this phenomenon, the simulator will be utilized for additional validation.

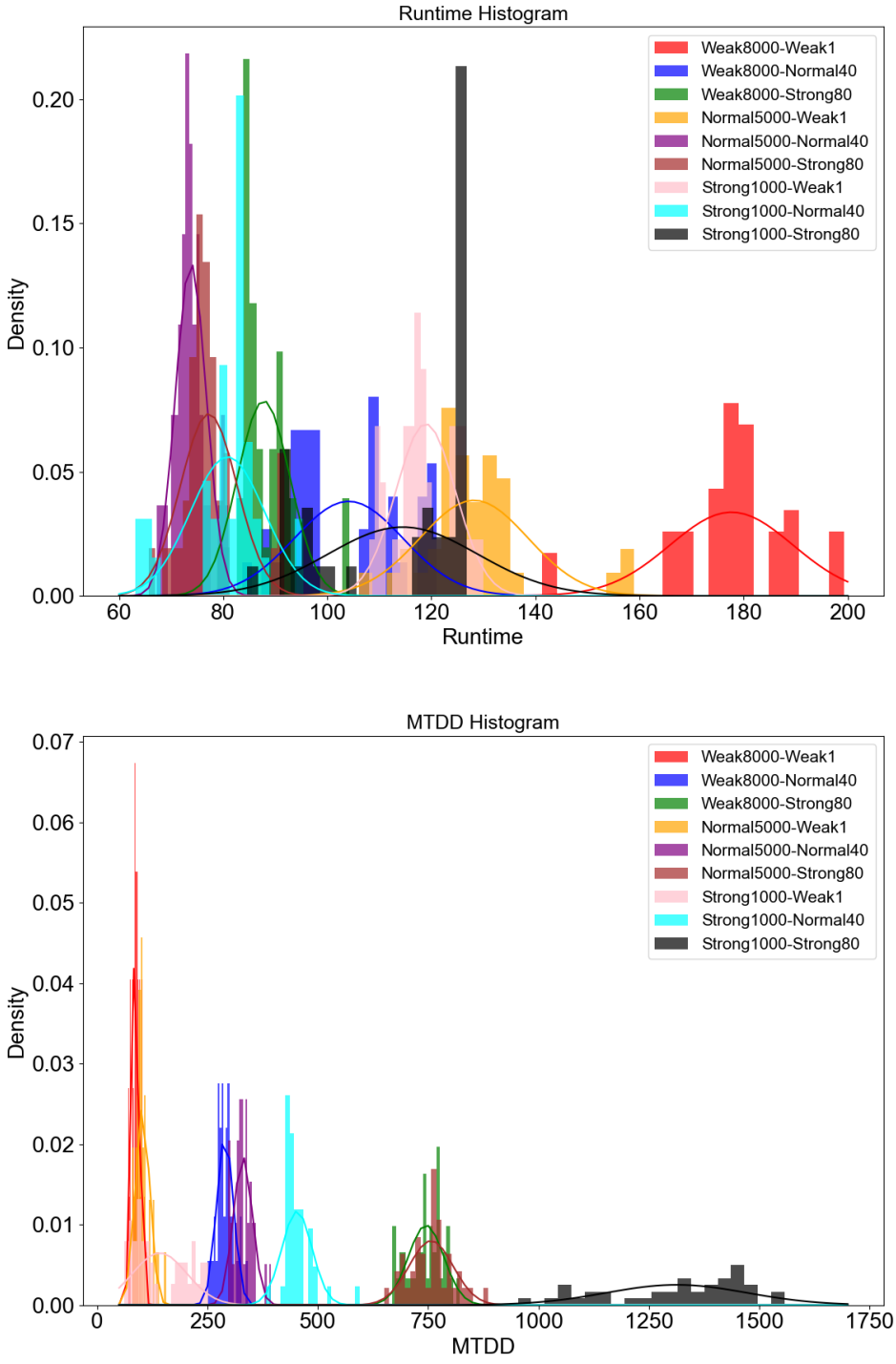
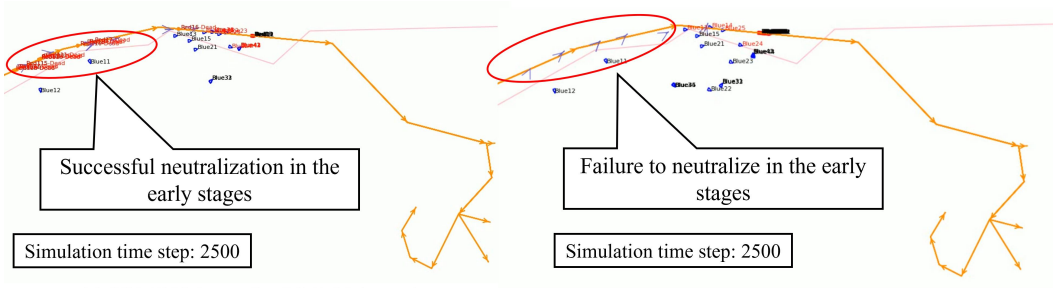


Figure 10. Density histogram of Runtime and MTDD. The colored lines represent normal distribution curves computed using the mean and variance of each dataset.



(a) SSD 5000 (normal) - SATA 80 (strong) (b) SSD 1000 (strong) - SATA 80 (strong)

Figure 11. Simulation visualization of 2 cases. (**a**) EFD normal—SATA strong case, (**b**) EFD strong—SATA strong case. The red circle indicates the area where the early-stage engagement occurs. In (**a**), within the red circle, there are SVs that were neutralized by USVs. However, in (**b**), within the same red circle, no USVs were neutralized.

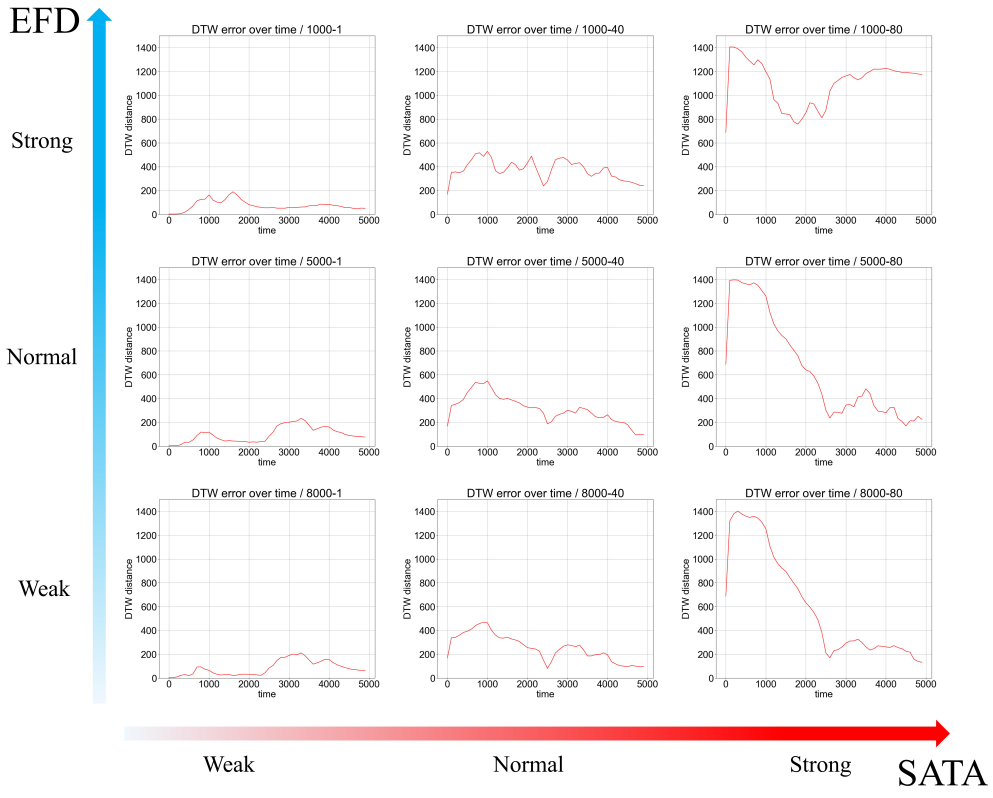


Figure 12. Trajectory Distance discrepancy over simulation time. While SATA has a significant impact on error, the influence of EFD is relatively small. However, when comparing the 'EFD strong-SATA strong' case to the 'EFD normal-SATA strong' case, it is evident that EFD influence under specific conditions.

5. Discussion

Although the proposed event filtering method demonstrates notable effectiveness in accelerating DEVS-based simulations, several limitations remain. First, the performance gain is highly dependent on the structural characteristics of each model. Different model configurations or interaction patterns may lead to varying trade-offs between runtime reduction and fidelity loss. Second, the effectiveness of the event filtering mechanism is sensitive to parameter tuning. In this study, parameters such as the Event Filtering Distance (EFD) and Sensor Acceleration Time Advance (SATA) were empirically determined, which may not be optimal for all scenarios. Developing an automated or adaptive parameter tuning strategy would further improve the robustness of the method. One possible direction

is to employ optimization or reinforcement learning-based techniques to dynamically identify suitable parameter values during simulation.

Beyond naval combat simulation, the proposed event-filtering approach can be extended to other domains that feature well-structured and human-engineered system architectures. Representative examples include manufacturing systems, smart factory operations, and industrial process simulations, where inter-model dependencies and data flows are explicitly defined. In such environments, dynamic coupling control can selectively reduce redundant communication events without compromising system functionality, thereby improving simulation efficiency. However, applying this method to black-box or purely data-driven systems would be challenging because their internal causal relationships are not directly observable. For these systems, additional interpretability or surrogate modeling mechanisms would be required to identify meaningful event dependencies before structural event filtering can be applied. This distinction highlights that the proposed approach is particularly suited to domains with transparent system logic and well-defined model interactions.

In addition, the proposed event filtering contributes to the stability and robustness of reinforcement learning by expanding the accessible learning domain. By accelerating data generation while maintaining consistent observation and reward semantics, the RL agent can explore a broader range of scenarios within limited computational resources. This improved diversity and density of experience enhances both sample efficiency and training stability, leading to faster and more reliable convergence of learning policies. Therefore, the proposed approach serves as an effective integration point between DSDEVS-based simulation acceleration and AI training pipelines.

The present study assessed simulation fidelity using the Mean Trajectory Distance Discrepancy (MTDD), which captures trajectory-level deviations between baseline and accelerated runs. This trajectory-based metric effectively quantifies micro-level behavioral consistency but does not fully represent higher-level outcomes such as engagement success or survival rate. Since the primary objective of this study was to examine how event filtering influences fine-grained motion behavior, MTDD was considered appropriate for this stage of analysis. Nevertheless, a more comprehensive evaluation of fidelity should account for both kinematic similarity and mission-level effectiveness. Future research will therefore extend the current framework by integrating trajectory-based and outcome-based indicators, enabling a multi-scale assessment that encompasses both continuous behavioral coherence and discrete engagement outcomes across various simulation domains.

6. Conclusions

This study proposed an event filtering method that improves the execution performance of a DEVS-based naval combat simulator by exploiting the structural flexibility of DSDEVSs. The method dynamically adjusts model couplings based on the importance of each event, reducing the computational load while maintaining simulation fidelity. In a USV naval combat scenario, where frequent sensing and communication causes long execution times, the proposed approach significantly shortened runtime without critical degradation of accuracy. The results confirm that DSDEVSs can be effectively used not only for structural modeling but also as a framework for performance optimization in complex event-driven simulations. By properly selecting parameters such as the Event Filtering Distance (EFD) and Sensor Acceleration Time Advance (SATA), the method provides a practical means to balance simulation speed and fidelity.

Author Contributions: Conceptualization, J.C.; methodology, J.C.; software, J.C.; validation, J.C. and I.-C.M.; formal analysis, J.C.; investigation, J.C.; resources, I.-C.M. and J.W.B.; data curation, J.C.; writing—original draft preparation, J.C.; writing—review and editing, I.-C.M. and J.W.B.; visualization, J.C.; supervision, I.-C.M. and J.W.B.; project administration, J.W.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: The authors would like to thank all colleagues who provided helpful feedback during the preparation of this work. This paper was supported by the Education and Research Promotion Program of KOREATECH in 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DES	Discrete Event System
DEVS	Discrete Event System Specification
DSDEVS	Dynamic Structure Discrete Event System Specification
MTDD	Mean Trajectory Distance Discrepancy
EFD	Event Filtering Distance
SATA	Sensing Acceleration Time Advance

Appendix A

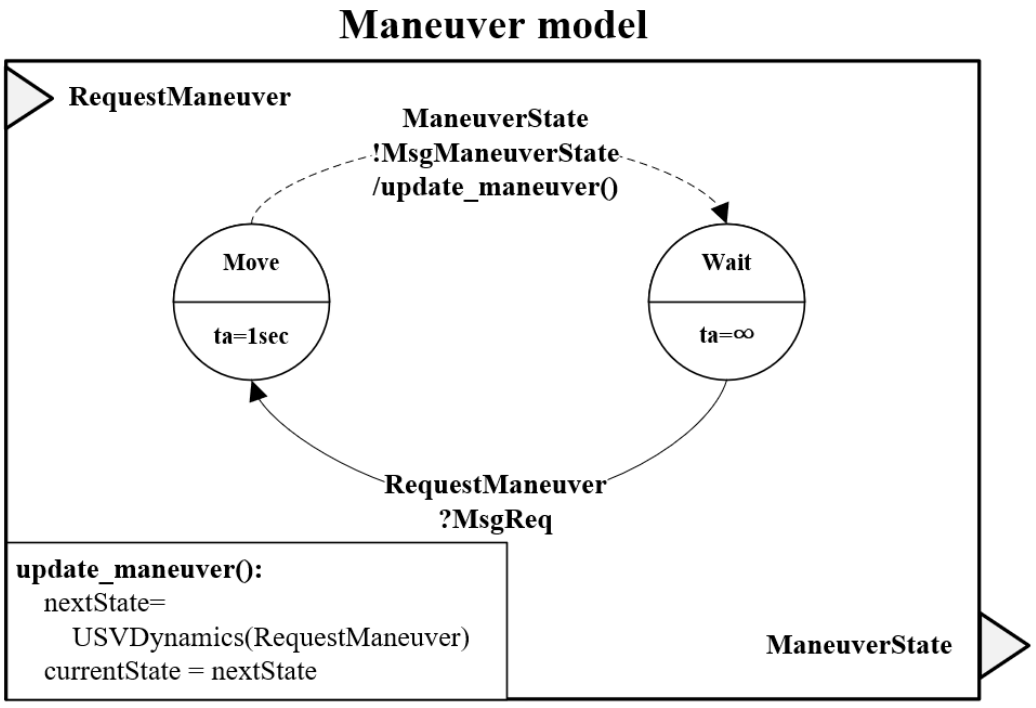


Figure A1. Diagram of Maneuver atomic model.

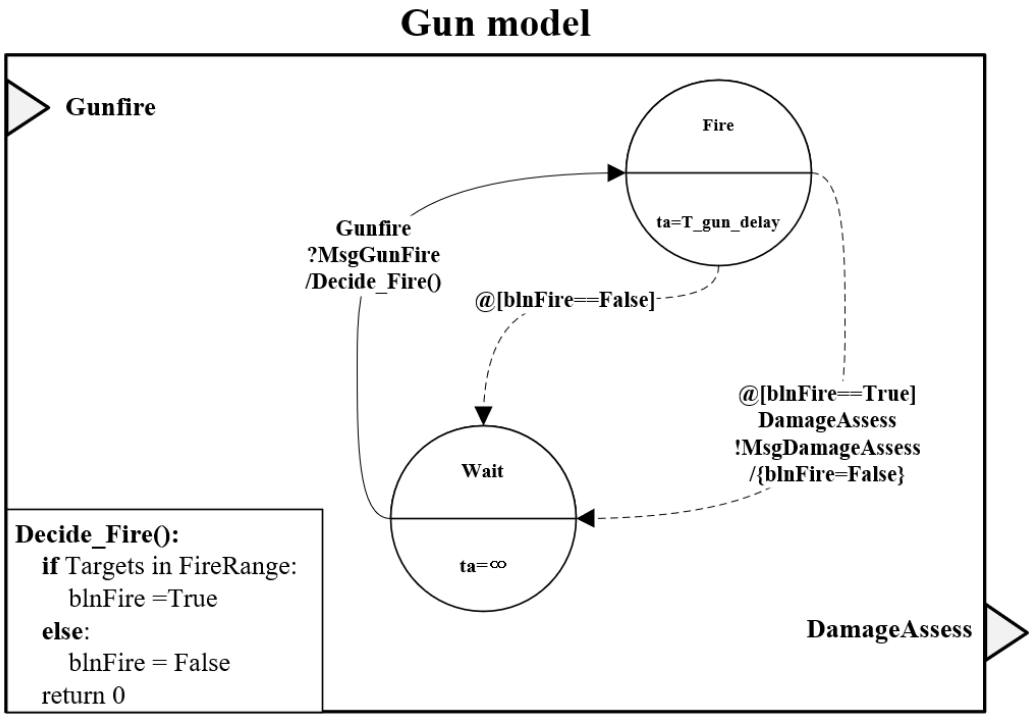


Figure A2. Diagram of Gun atomic model.

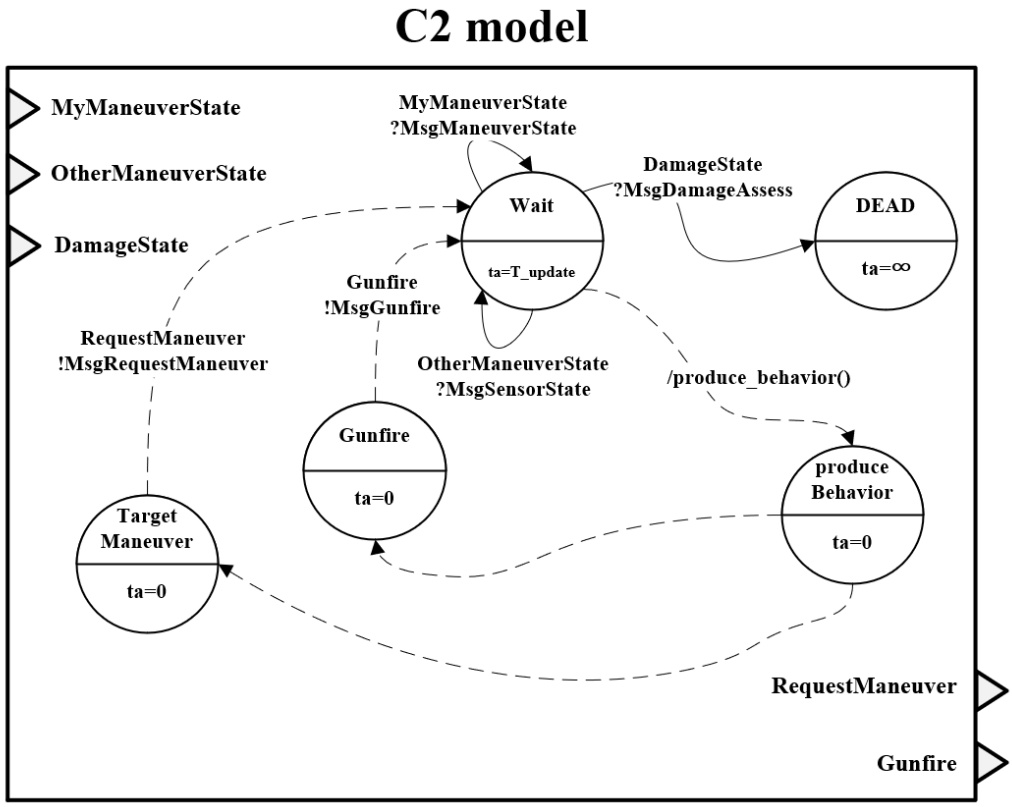


Figure A3. Diagram of C2 (Command & Control) atomic model.

References

1. Zeigler, B.P.; Praehofer, H.; Kim, T.G. *Theory of Modeling and Simulation*; Academic Press: San Diego, CA, USA, 2000.
2. Barros, F.J. Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modeling and Simulation. In Proceedings of the 27th Conference on Winter Simulation (WSC 1995), Arlington, VA, USA, 3–6 December 1995; pp. 781–785.

3. Chow, A.C.H.; Zeigler, B.P. Parallel DEVS: A Parallel, Hierarchical, Modular Modeling Formalism. In Proceedings of the Winter Simulation Conference (WSC 1994), Orlando, FL, USA, 11–14 December 1994; pp. 716–722.
4. Zeigler, B.P. Distributed Supply Chain Simulation in a DEVS/CORBA Execution Environment. In Proceedings of the 1999 Winter Simulation Conference (WSC), Phoenix, AZ, USA, 12–15 October 1999; pp. 1540–1545.
5. Zeigler, B.P.; Nutaro, J.; Seo, C. What's the Best Possible Speedup Achievable in Distributed Simulation: Amdahl's Law Reconstructed. In Proceedings of the 2008 Spring Simulation Multiconference (SpringSim), Ottawa, ON, Canada, 14–17 April 2008; pp. 123–130.
6. Trabes, G.G.; Wainer, G.A.; Gil-Costa, V. A Parallel Algorithm to Accelerate DEVS Simulations in Shared Memory Architectures. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 1609–1620. [CrossRef]
7. Liu, Q.; Wainer, G. Exploring Multi-Grained Parallelism in Compute-Intensive DEVS Simulations. In Proceedings of the 2010 ACM/IEEE/SCS 24th Workshop on Principles of Advanced and Distributed Simulation (PADS), Atlanta, GA, USA, 17–19 May 2010; pp. 1–8.
8. Himmelspach, J.; Ewald, R.; Leye, S.; Uhrmacher, A.M. Parallel and Distributed Simulation of Parallel DEVS Models. In Proceedings of the 2007 Spring Simulation Multiconference (SpringSim), Norfolk, VA, USA, 25–29 March 2007; pp. 249–256.
9. Cárdenas, R.; Henares, K.; Arroba, P.; Wainer, G.; Risco-Martín, J.L. A DEVS Simulation Algorithm Based on Shared Memory for Enhancing Performance. In Proceedings of the 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 14–18 December 2020; pp. 2336–2347.
10. Li, B.; Hu, K. A Formal Description Specification for Multi-Resolution Modeling (MRM) Based on DEVS Formalism. In *AI, Simulation, and Planning in High Autonomy Systems*; Springer: Jeju Island, Republic of Korea, 2004; pp. 285–294.
11. Li, B. A Formal Description Specification for Multi-Resolution Modeling Based on DEVS Formalism and Its Applications. *J. Def. Model. Simul.* **2007**, *4*, 229–251.
12. Wang, J.; Tropper, C. Optimizing Time Warp Simulation with Reinforcement Learning Techniques. In Proceedings of the 2007 Winter Simulation Conference (WSC), Washington, DC, USA, 9–12 December 2007; pp. 577–584.
13. Coopmans, T.; Knegjens, R.; Dahlberg, A.; Maier, D.; Nijsten, L.; de Oliveira Filho, J.; Papendrecht, M.; Rabbie, J.; Rozpędek, F.; Skrzypczyk, M.; et al. NetSquid: A NETwork Simulator for QUantum Information using Discrete Events. *Commun. Phys.* **2021**, *4*, 164. [CrossRef]
14. Gao, K.; Chen, L.; Li, D.; Liu, V.; Wang, X.; Zhang, R.; Lu, L. DONS: Fast and Affordable Discrete-Event Network Simulation with Automatic Parallelization. In Proceedings of the ACM SIGCOMM 2023, New York, NY, USA, 6–10 August 2023; pp. 167–181.
15. Hong, L.J.; Zhang, X. Surrogate-Based Simulation Optimization. In *Emerging Optimization Methods and Modeling Techniques with Applications*; Carlsson, J.G., Ed.; INFORMS Tutorials in Operations Research: Catonsville, MD, USA, 2021; pp. 287–311.
16. Zhang, Q.; Wang, Y.; Zhou, A. Surrogate-Assisted Evolutionary Algorithms for High-Cost Problems: A Survey. *IEEE Trans. Evol. Comput.* **2021**, *25*, 189–208.
17. Saadawi, H.; Wainer, G. DEVS Execution Acceleration with Machine Learning. In Proceedings of the Symposium on Theory of Modeling and Simulation (TMS), Pasadena, CA, USA, 3–6 April 2016; pp. 1–8.
18. Capocchi, L.; Santucci, J.F. Discrete Event Modeling and Simulation for Reinforcement Learning System Design. *Information* **2022**, *13*, 121. [CrossRef]
19. *IEEE Std 1516.2-2010*; IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)–Object Model Template Specification. IEEE: Piscataway, NJ, USA, 2010.
20. Tan, G.; Xu, L.; Moradi, F.; Taylor, S. An agent-based DDM for High Level Architecture. In Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS 2001), Piscataway, NJ, USA, 15–18 May 2001; pp. 75–82. Available online: https://ieeexplore.ieee.org/abstract/document/924623 (accessed on 29 October 2025). [CrossRef]
21. Sui, B.; Zhao, Y.; Chen, Y.; Liu, C.; Tang, J. Prescribed-Time Trajectory Tracking Control for Unmanned Surface Vessels with Prescribed Performance Considering Marine Environmental Interferences and Unmodeled Dynamics. *J. Mar. Sci. Eng.* **2024**, *12*, 1380. [CrossRef]
22. Li, J.; Xu, H.; Yu, C.; Liu, Z. Prescribed Performance Path Following Control of USVs via an Output-Based Threshold Rule. *Ocean Eng.* **2023**, *284*, 115317. [CrossRef]
23. Zhang, W.; Sun, Z.; Wang, P.; Liu, Q. Prescribed Performance Control for USV–UAV via a Robust Bounded Compensating Technique. *Appl. Ocean Res.* **2025**, *145*, 103812. [CrossRef]
24. Müller, M. Dynamic Time Warping. In *Information Retrieval for Music and Motion*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 69–84.