A Modular Execution Architecture for Robust Multi-Robot Planning and Acting in Trans-Media Environments

Virgile de la Rochefoucauld, Photchara Ratsamee, Simon Lacroix, Félix Ingrand, Yuki Uranishi

A Modular Execution Architecture for Robust Multi-Robot Planning and Acting in Trans-Media Environments

Virgile de La Rochefoucauld[1,3], Photchara Ratsamee[2], Simon Lacroix*[1], Félix Ingrand[1], and Yuki Uranishi[3]

[1]LAAS-CNRS, INSA, University of Toulouse,Toulouse, France
[2]Faculty of Engineering Science, Department of Mechanical Engineering, Kansai University, Osaka, Japan
[3]Cybermedia Education Division, D3 Center, The University of Osaka, Osaka, Japan

Abstract

In complex missions involving heterogeneous multi-robot teams, especially in trans-media systems that operate across environments such as air and water, robust execution frameworks must ensure both temporal coherence and resilience to uncertainty. These challenges stem from the need to manage dynamic mode transitions, closely linked inter-agent actions, and execution-time failures. This paper introduces the Adaptive and Modular Architecture (AMA) Execution and Planning components. AMA-EXEC is a distributed execution framework designed to enable coherent, fault-tolerant mission execution in such conditions. AMA-EXEC uses the plans produced by AMA-PLAN, a PDDL-based planning framework. AMA-EXEC incorporates Simple Temporal Networks (STNs) to facilitate temporal reasoning; modular Behavior Trees (BTs) to enable distributed execution; and runtime monitoring mechanisms to categorize failures, propagate delays, and execute partial replanning. In contrast to centralized, monolithic systems, AMA-EXEC organizes execution around collaborative robot teams and leverages real-time feedback to maintain synchronization and temporal alignment under disturbances. We validate the framework on simulated trans-media missions, encompassing concurrent actions and a range of failure scenarios. Findings indicate enhanced execution continuity, applicability to more complex missions, and robust failure recovery in comparison to baseline methodologies. AMA-EXEC modularity and generalizability render it suitable for a wide range of applications, including environmental monitoring, distributed exploration, and search-and-rescue operations.

*Simon Lacroix has passed away recently, however, his contributions to the presented work are significant and we decided to keep him as a co-author of the paper.

1 Introduction

The deployment of autonomous multi-robot systems has become increasingly central to missions in exploration, monitoring, infrastructure inspection, and disaster response. Recent works on homogeneous multi-robot systems [Reina et al., 2015, Schweim et al., 2024] emphasized scalability and robustness through coordination and swarm strategies, while heterogeneous teams [Carreno et al., 2020, Vasilijevic et al., 2017, Shkurti et al., 2012] introduced role specialization and capability distribution. More recently, trans-media robot teams [Yao et al., 2023, Pedroso et al., 2022, Tan and Chen, 2021, 2019], capable of operating across different physical domains, such as air, water, and land, have been developed and studied, offering new operational possibilities and solutions for complex missions spanning multiple media. However, this increased functionality introduces significant execution challenges: agents must reason about medium-specific constraints, coordinate mode transitions, and manage tight temporal and sequential dependencies between inter-agent actions. In such dynamic and uncertain environments, actions are more prone to be delayed, fail, or result in partial execution. Consequently, plans must adapt at runtime through failure monitoring and synchronization.

The planning of operations for multiple robots has been a subject of interest and study for some time. The formalization of the Multi-Robot Task Allocation (MRTA) [Gerkey and Matarić, 2004] planning problem, which is central to the community, has been a focal point of research. This formalism introduces challenges, definitions, and recurring models of various problems faced by robots during task allocation. While symbolic planning approaches based on the Planning Domain Definition Language (PDDL) [Ghallab et al., 1998] offer a powerful and general framework for describing such high-level mission objectives, they often face limitations in complex, dynamic deployment. A prominent ROS 2-native example is PlanSys2 [Martín et al., 2021], which builds on PDDL and incorporates BT-based execution [Colledanchise and Ögren, 2018] for practical deployment. However, such systems face challenges in dynamic, real-world settings where delays, failures, and communication issues are common.

Practical deployment of trans-media robots also hinges on lower-level autonomy layers such as localization, navigation, and medium-specific sensing, which introduce their own uncertainties in the field. For instance, surface-level SLAM can suffer from perceptual aliasing and dynamic disturbances [Ratsamee et al., 2023], while underwater and aerial operations face sensing gaps and costly medium transitions. These challenges highlight that even with high-quality plans, reliable execution requires an architecture capable of absorbing delays, partial failures, and synchronization drift at runtime.

In this paper, we present AMA, an **A**daptive and **M**odular **A**rchitecture for robust multi-robot planning and execution in complex environments designed to provide this supervisory robustness on top of existing lower-level modules. It consists of two main interleaved components:

- AMA-PLAN, which restructures complex missions into temporally annotated, execution-ready symbolic plans using spatial clustering, coalition formation, meta-task allocation, and dependencies compatible with STNs [Dechter et al., 1991].

- AMA-EXEC, a distributed execution framework based on PlanSys2, that supervises robot teams via modular distributed BTs, enforces symbolic-temporal constraints using the STN-

CONTROLLER, and handles localized failures through the Distributed Execution Manager (DEM) without requiring full replanning.

AMA provides symbolic planning and reactive execution by combining modularity, parallel execution, temporal reasoning, and runtime fault analysis. It enables reliable multi-robot coordination, even under asynchronous delays and execution failures.

To validate our system, we simulate trans-media missions involving multiple aerial-aquatic robots executing collaborative actions across distributed sites. We demonstrate that AMA improves mission continuity, synchronization robustness, parallelization, and recovery efficiency compared to baseline approaches.

This paper is structured as follows. Section 2 reviews related work on multi-robot planning-acting architectures and situates AMA within this context. Section 3 provides an overview of the AMA-EXEC architecture and illustrates its execution flow across the main modules. Section 4 formulates the planning problem and explains how AMA-PLAN, building on our previous work [De La Rochefoucauld et al., 2024] and its extensions, addresses its inherent complexity. Section 5 describes the execution logic, including temporal synchronization and failure recovery handled by the STN-CONTROLLER and the DEM. Section 6 presents the simulation setup, experimental protocol, and quantitative results. Finally, Section 7 concludes the paper and discusses perspectives for future work, including lessons from preliminary real-world deployment and possible extensions beyond trans-media missions.

2 Related Work and Proposed Approach

This section reviews the broader context of multi-robot planning and execution systems, highlighting existing symbolic and temporal frameworks. We first examine the recent trends in joint planning and execution approaches, and frameworks dealing with their deployment in multi-robot domains. We then consider BT-based systems and frameworks offering ROS 2 integration. Finally, we position AMA relative to these works, highlighting how it bridges symbolic planning with runtime execution robustness.

There is a growing consensus in the planning community that robotic autonomy requires tight coupling between planning, acting, monitoring, and failure handling [Ghallab et al., 2016]. Several frameworks attempt to achieve this through hybrid architectures, operational models [Patra et al., 2019], or reactive control loops. PDDL [Ghallab et al., 1998] provides a syntax for expressing planning problems in a structured way and supports various extensions beyond classical STRIPS planning. It has since become the de facto standard for symbolic planning research and has been widely adopted in robotic frameworks through various extensions such as [Brenner, 2003, Kovacs, 2012] for multi-agent problems.

Moreover, the integration of symbolic planning into robotic systems became more widely adopted through *ROSPlan* [Cashmore et al., 2015], a ROS 1-native framework allowing plan generation from PDDL models and execution via a centralized dispatcher. ROSPlan provided foundational tools for integrating automated planning with real-robot execution. However, its capabilities were limited to single-agent or tightly synchronized multi-robot scenarios, without support for real-time temporal reasoning, distributed team management, or reactive recovery.

Other frameworks proposed to integrate acting and planning in robotics systems, e.g. with temporal reasoning [Ingrand et al., 2007, Finzi et al., 2004, Umbrico et al., 2017]), but for single-robot domains. Beyond ROS-based architectures, several operational models have aimed to tightly integrate symbolic planning and acting. Systems such as the Refinement Acting Engine (RAE) [Patra et al., 2019, 2021] and OMPAS [Turi and Bit-Monnot, 2022] introduce hierarchical acting models that support deliberative reasoning, reactive adaptation, and skill-level abstraction. OMPAS, in particular, introduces decentralized policy search, enabling robots to autonomously refine and execute plans within constraints. However, these systems often rely on domain-specific operational primitives and incur a high computational cost (e.g., Monte Carlo Tree Search in [Bramblett et al., 2024]). Additionally, they are not optimized for temporal synchronization across distributed teams.

Earlier studies have investigated the issue of execution supervision in distributed settings. The *Plan Manager* [Joyeux et al., 2009] coordinated concurrent task execution under global plans, and *Stedl et al.* [Stedl and Williams, 2005] proposed distributed execution of temporally flexible plans. *dMARS* [D'Inverno et al., 2004] and [Brenner and Nebel, 2009], addressed interleaving planning and acting under uncertainty for continual planning, while [Di Rocco et al., 2013] studied plan repair for late or delayed robots in dynamic goal settings. [Belbachir et al., 2012] proposed a cooperative architecture for coordinating AUVs, emphasizing the complexity of execution under heterogeneous roles and media. These systems anticipated many of the challenges that are currently being targeted, yet lacked simplified modular execution backends or robotic systems integration.

Beyond symbolic and temporal execution frameworks, a substantial body of research addresses multi-agent coordination at the continuous-control layer. Foundational surveys on consensus and cooperative control [Olfati-Saber et al., 2007, Ren and Beard, 2010] established distributed protocols for state agreement, formation keeping, and information fusion. More recent works have extended these ideas to challenging settings: predefined/fixed-time stabilization for fast, bounded convergence [Polyakov, 2012]; adaptive and neural-network–assisted controllers for uncertain nonlinear agents [Li et al., 2025b]; fuzzy adaptive optimal consensus under switching or adversarial conditions [Wang et al., 2025]; event-triggered coordination with intermittent actuator faults [Wang et al., 2020]; and asynchronous cooperation–competition networks [Li et al., 2025a].

These contributions focus on ensuring stability, convergence, and resilience of the underlying dynamics, and they typically leverage control theory or learning-based approaches. Our work stays within classical automated planning: AMA consumes symbolic goals expressed in PDDL, restructures them through decomposition and temporal reasoning, and supervises their execution with explicit constraint tracking. In this sense, consensus and formation controllers are complementary to AMA: they provide robustness at the control and coordination levels, while AMA operates at the high level decisional layer to ensure symbolic and temporal coherence and localize recovery when failures occur.

Recently, BTs have gained popularity for their modularity, reactivity, and hierarchical execution logic. In [Colledanchise and Ögren, 2018, Colledanchise et al., 2019], the authors investigated the potential of BTs for reactive planning, acting, and AI, proposing blends with deliberative systems. BTs have since been applied to multi-robot systems for the purpose of distributed

coordination [Jeon et al., 2022], task transfer [Ghzouli et al., 2023], and flexible mission execution [Iovino et al., 2022]. BT-based systems, despite their elegance and adaptability, frequently remain centralized, lacking both symbolic supervision and runtime temporal constraint checking.

PlanSys2 [Martín et al., 2021] has been developed to address the limitations of ROS 1 in the context of multi-robot systems and to transition to the more recent ROS 2. The system maintained symbolic modeling via PDDL while incorporating a BT-based execution backend [Martín et al., 2021], thereby enabling modular and hierarchical action execution. However, PlanSys2 still assumes centralized execution, lacks inter-agent symbolic dependency tracking and offers no built-in mechanisms for temporal supervision or adaptive failure recovery.

In their recent work [Zapf et al., 2024], the PlanSys2 team proposes a BT-from-STN approach, synthesizing BTs directly from STNs by encoding symbolic actions and their temporal constraints as BT nodes. This integration allows BTs to mirror the temporal structure of the planning phase. However, the system:

- Builds fixed temporal-checking nodes during BT construction.
- Provides no mechanism to handle drift, latency, or constraint violations at runtime.
- Establishes a heavy PDDL $\rightarrow$ STN $\rightarrow$ BT structure which may further increase building time in complex multi-robot planning.

To address execution-time robustness and build-time overhead, AMA-EXEC incorporates several innovations for executing symbolic plans. First, a dedicated STN-CONTROLLER supervises execution by maintaining a live precedence graph of symbolic and temporal dependencies. In contrast to BT-from-STN approaches, which encode temporal structure statically at build time, our system performs active monitoring: it continuously tracks action progression in the STN, detects discrepancies at upcoming bottlenecks, and enforces constraints through delay injection when needed.

Second, AMA-EXEC decentralizes plan execution: AMA-PLAN decomposes missions into constrained subproblems, which are executed by modular, per-team BT executors so that distributed robot groups operate asynchronously under unified symbolic constraints. This modularization preserves standard PDDL plans while embedding real-time supervision and constraint propagation directly into execution, bridging centralized planning with dynamic, distributed, failure-prone environments. Compared to operational-model executives (e.g., RAE/OMPAS), which execute over richer domain-specific skill models at the cost of heavier online search, AMA-EXEC deliberately retains PDDL plans and limits online search via per-team decomposition and early-split caps.

Finally, the DEM coordinates recovery via constraint-aware reallocation and failure handling, ensuring that replanning remains localized to the minimum affected scope. Together, these modules establish a flexible, runtime-supervised execution pipeline capable of managing complex multi-robot missions. Rather than replacing existing paradigms, our approach integrates the structural clarity of symbolic plan decomposition, the modular robustness of PlanSys2-style BT execution, while extending it with temporal responsiveness through STN-based delay propagation. BT-from-STN methods compile temporal structure at build time, whereas AMA-EXEC

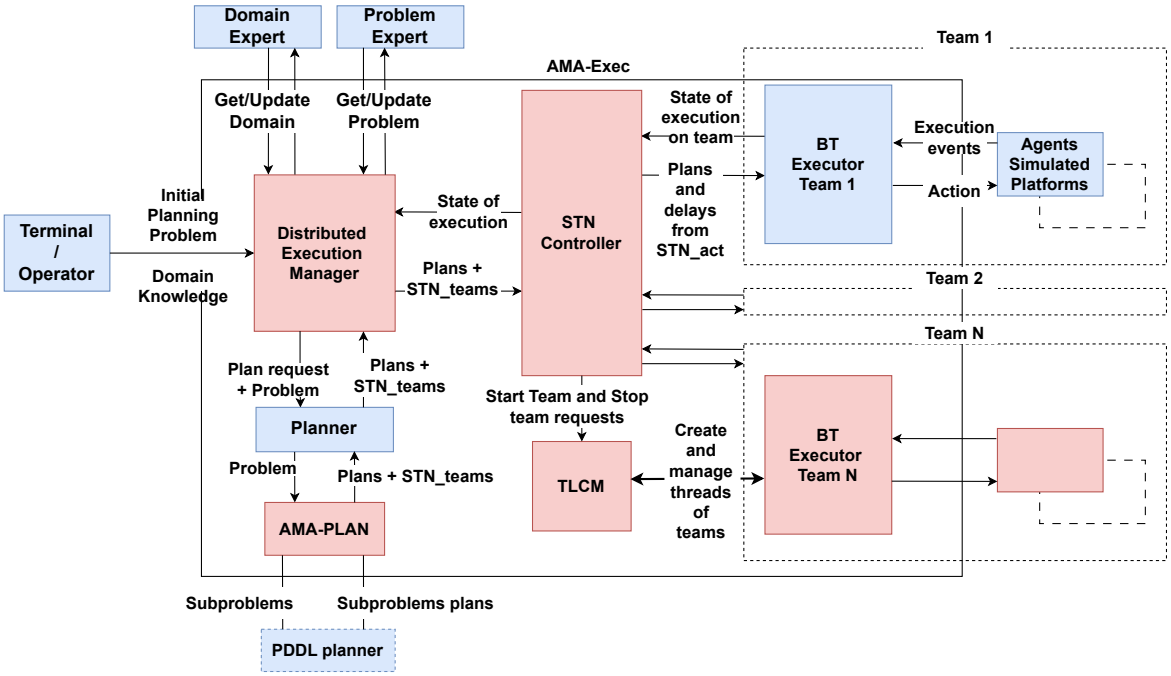


Figure 1: High-level overview of AMA-EXEC execution components (in red) and PlanSys2 original components (in blue).

enforces it dynamically at runtime through distributed STN_{act}. Operational-model executives such as RAE and OMPAS provide richer online reasoning with different scalability trade-offs, while AMA-EXEC emphasizes scoped recovery over standard, generalizable PDDL plans.

3 System Architecture and Design Principles

This section introduces the architectural foundations of AMA-EXEC, providing a high-level view of its modular components, their roles, and the overall mission execution pipeline. The architecture extends the PlanSys2 framework to support modularity, scalability, and temporal robustness in complex multi-robot missions.

3.1 Motivation and Architecture Overview

In Figure 1 we present an overview of the AMA-EXEC framework, incorporating the original PlanSys2 architecture components (in blue boxes) with the extensions presented in this paper (in light red). Although PlanSys2 offers PDDL-based planning and BT-based execution, it lacks support for temporal supervision, distributed team management, and adaptive failure recovery presented in AMA-EXEC.

The AMA-EXEC execution framework is grounded in symbolic plans produced by AMA-PLAN, an execution-ready enhancement of the work presented in [De La Rochefoucauld et al.,

2024]. The symbolic plans produced by AMA-PLAN are considered *execution-ready* as they are temporally annotated, robot-state consistent across subproblems, and embedded in a structured two-layer temporal model. This includes a team-level STN (STN_{teams}), which encodes inter-team precedence and synchronization constraints, and individual action-level STNs (STN_{act}) for each robot team, which capture intra-team PDDL plan temporal dependencies. These plans result from spatio-temporal clustering, task allocation, and mission decomposition performed by AMA-PLAN. Once generated, AMA-EXEC ensures that these high-level plans are executed in real-time, even in the case of temporal drift, action delays, or failures.

To address the limitations of PlanSys2, AMA-EXEC introduces additional modules, each responsible for a specific layer in the pipeline. Each component manages its responsibilities independently, from symbolic coordination to reactive control, with ROS 2 ensuring safe concurrency, threading, and isolation across teams. AMA-EXEC is designed to support the robust execution of complex multi-robot missions. The next subsection details the core modules that compose this architecture and their specific responsibilities.

3.2 Modular Component Responsibilities

The AMA-EXEC framework is designed around five core modules responsible for different layers of mission execution and monitoring:

- **AMA-PLAN**: This toolbox decomposes the mission into symbolic subproblems by performing spatial clustering, coalition formation, and meta-task allocation. It generates inter-team dependency structures for downstream execution into STN_{teams} precedence graph, and solves each subproblem through a chosen PDDL solver.

- **STN-CONTROLLER**: This component enforces temporal constraints between and within teams, by monitoring action-level execution and injecting delays when necessary through the proper STN_{act}. It ensures that dependent actions (e.g., concurrent actions) are executed/finished in the correct order and with appropriate synchronization margins. It also determines when new teams can be launched based on progression of the mission execution in the STN_{teams}.

- **Distributed Execution Manager (DEM):** Acting as the mission-level controller, the DEM receives symbolic plans and STN_{teams} from AMA-PLAN, parses the result into executable structures, starts and monitors STN-CONTROLLER and its execution feedback, and handles partial replanning or recovery in case of failure.

- **Team Lifecycle Manager (TLCM):** The TLCM is responsible for the lifecycle management of team execution, their BT Executors and simulated robots. It handles the launch, supervision, and termination of team's environment, allowing dynamic team creation and removal at runtime.

- **Multi-BT Executors:** These BT-based execution nodes are derived from PlanSys2 and extended to operate in ROS 2 namespaces and dedicated threads, allowing parallel execution.

Module	Role	Inputs	Outputs
AMA-PLAN	Mission problem decomposition and symbolic plan generation	World data, domain, problem	Teams symbolic plans + STN_{teams}
STN-CONTROLLER	Temporal supervision, execution control	Team plans, STN_{teams}, execution feedback	Start requests, delays, temporal adjustments and controls
DEM	Plans parsing, STN-CONTROLLER init., failure analysis	AMA-PLAN plans, STN-CONTROLLER feedback	Parsed team plans, replanning triggers, STN-CONTROLLER updates
TLCM	Team lifecycle control: robot/team node spawning	Start/stop requests from STN-CONTROLLER	BT Executor nodes, simulated robot threads
Multi-BT Executors	Team execution Action-level control	Team-assigned symbolic plan	Action status, feedback topics, mission completion flags

Table 1: AMA-EXEC component roles, inputs, and outputs in the distributed execution pipeline.

Each executor controls the actions of a specific team, reacting to feedback and executing symbolic instructions while remaining decoupled from other teams' execution flows.

This modular organization enables AMA-EXEC to run distributed missions with asynchronous team behaviors, localized failure response, and high concurrency while limiting bottlenecks. These modules operate in coordination across planning, temporal control, and distributed execution. Their roles, inputs, and outputs are summarized in Table 1, which provides a modular overview of the execution pipeline. The next subsection builds on this by detailing how these components interact in practice, describing the flow of information from symbolic mission planning to distributed execution.

3.3 Execution Flow and Module Interactions

This subsection details the information flow between AMA-EXEC components, from symbolic mission planning to distributed execution. It reflects the modular startup, sequential dispatching, and coordination process enabled by ROS 2 threading and PlanSys2 integration.

AMA-EXEC initializes, monitors, and adapts execution across robot teams using its modular design. The execution flow is as follows:

1. At launch time, core components are brought online:
   - The **PlanSys2 core**, including the domain/problem experts and planner node;
   - The **DEM**, which monitors and orchestrates planning and execution;
   - The **TLCM**, responsible for spawning and managing the teams' environments.
2. Once the problem and domain definitions are available (either from launch or through terminal requests), the DEM issues a planning request via PlanSys2's standard interface.
3. AMA-PLAN module receives this request and performs decomposition of the mission, generating symbolic plans per robot team through the PDDL planner along with a set of sequential dependency links between them.

4. Upon receiving the team plans and their temporal dependencies, the DEM initializes the STN-CONTROLLER thread. Team plans and dependency links in STN_{teams} are passed to the controller, which constructs the STN_{act} for each team, capturing dependencies within each team.

5. From this point, the STN-CONTROLLER supervises execution using the STN_{teams} graph:

    - It launches teams as their dependencies are satisfied, coordinating with the TLCM to start BT executors;
    - It publishes symbolic plans to executors and monitors progress through execution feedback;
    - Execution proceeds asynchronously across teams, with the controller ensuring symbolic-temporal constraints are respected.

When a team succeeds, the STN-CONTROLLER removes its executor and action nodes through a request to TLCM, freeing resources. It then activates the next plan in the STN_{teams} whose dependencies are now satisfied. This process repeats until all symbolic plans are completed or a failure prevents continuation.

Throughout execution, the DEM remains active in the background, receiving execution status from the STN-CONTROLLER. In the event of a symbolic failure or plan-level inconsistency, the DEM evaluates impact across teams and provides a failure report to AMA-PLAN, which then determines the appropriate level of replanning.

This execution flow establishes a tightly coordinated pipeline that connects the symbolic planning with modular execution. Each module activates in a defined sequence, enabling multi-robot teams to operate simultaneously while maintaining global synchronization. At the action level, these interactions rely on PlanSys2's BT executors, which AMA-EXEC extends for distributed and modular deployment, as detailed below.

3.4 Action-Level Execution: BT Executors and PlanSys2 Integration

At a lower level, each robot team executes its symbolic plan using a dedicated BT executor node. AMA-EXEC preserves PlanSys2's BT-based execution core while extending its deployment for modular use, specifically:

- **Action namespace scoping:** Each BT executor is operated within a dedicated ROS 2 namespace corresponding to its team and robot's actions, preventing cross-team topic interference;

- **Dynamic team lifecycle management:** Execution environments, including BT Executors, robots and action nodes are instantiated and managed by the TLCM dynamically based on STN-CONTROLLER requests;

- **Feedback multiplexing:** Each team environment exposes multiple feedback channels reporting BT execution status, robot states, and action node outcomes.

Together, these design choices make AMA-EXEC suitable for real-world deployment in multi-robot missions with dynamic environmental constraints, failure modes, and inter-team dependencies. Its structured separation of responsibilities allows:

- Independent initialization, execution, and shutdown of robot teams;
- Flexible replanning in response to failure, based on localized impact;
- Real-time enforcement of temporal constraints without centralized blocking;
- Parallel team execution without resource contention.

Further details on how AMA-EXEC ensures temporal alignment, delay injection, and symbolic-temporal consistency enforcement during execution are provided in Section 5.

4 Problem Modeling and Planning

This section formalizes the planning problem addressed by our architecture and details the AMA-PLAN framework used for mission decomposition, which provides plans to AMA-EXEC. It summarizes the planning model and outlines the modular extensions introduced since our initial work [De La Rochefoucauld et al., 2024].

4.1 Planning Problem Definition

We define the planning problem addressed in our use case as the generation of temporally constrained, multi-agent plans for a team of trans-media robots with dynamic capabilities. The robots $\mathscr{R} = \{r_1, \ldots, r_n\}$ are tasked with visiting and operating on a set of points of interest (PoIs) $P = \{p_1, \ldots, p_m\}$ contained in a set of geolocated sites $\mathscr{S} = \{s_1, \ldots, s_k\}$ (we denote $P_s \subseteq P$ the set of PoIs located at s) under medium-specific constraints as presented in Figure 2. P is divided into 3 types of PoIs with different characteristics: the aquatic sampling points P_{spl}, the surface transition points P_{tr} and the aerial observation points P_{obs} , with $P = P_{spl} \bigcup P_{tr} \bigcup P_{obs}$. Each robot can operate in one or more media, denoted by $M = \{\text{aerial}, \text{aquatic}, \text{terrestrial}\}$. Transition between media is allowed exclusively through the designated P_{tr} transition points. The actions encompass navigation, observation, data sampling, transmission, and medium switching. These actions are modeled using PDDL2.1 [Fox and Long, 2003] durative actions with temporal and concurrency constraints.

4.1.1 Action Set Overview.

To illustrate the mission representation, the symbolic action types are detailed below:

- `observe`: One or several robots view a site's transition points from its aerial observation point;
- `navigate`: Move between PoIs within the same medium;

- `switch`: Transition from one medium to another at a surface transition point;
- `translate_data`: Relay underwater data at the surface for uplink to base;
- `sample`: Perform aquatic data collection (requires concurrent data conversion);
- `takeoff/land`: Change between terrestrial and aerial states at the base.

We associate a cost based on action durations:

- `sample`, `translate_data`, `takeoff` and `land` actions have fixed costs, `observe` cost depends on the number of robots involved.
- `switch` actions have two fixed costs, determined by the direction of the transition (Aquatic↔Aerial)
- `navigate` actions have a cost defined by the distances $D(p_u, p_v)$ between the PoIs (p_u, p_v) and the medium m_r in which they are executed:

$$C_{\text{nav}}(r, p_u, p_v) = \frac{D(p_u, p_v)}{V(m_r)}. \tag{1}$$

  where $V(m_r)$ is the velocity of the robot r in the medium m_r and $D(\cdot,\cdot)$ is the Euclidean distance.

All fixed costs and action choices are grounded in platform specifications, literature surveys, and field observations from trans-media deployments (e.g., asymmetric Aquatic↔Aerial transition costs; collaborative surface relaying to avoid radio-frequency loss), and the same parameters drive AMA-EXEC's execution constraints (durations, margins, failure classes).

4.1.2 Illustrative Action: Switch.

Each action class defines conditions, durations, and resource constraints. As an example, the switching action is developed in detail below[1]:
Switch: This action symbolizes a medium transition from m_k to m_l for a robot r and can only be executed at a transition point p, where $m_k \neq m_l$ and both are supported by point p. The set is defined as follows:

$$\begin{aligned} A_{sw} = \{ & a_{sw}(r, p, m_k, m_l, s) \\ & \mid r \in \mathscr{R},\ p \in P_{tr} \cap P_s, m_k, m_l \in M \text{ with } m_k \neq m_l \} \end{aligned}$$

It should be noted that this action can only be executed subsequent to the observation of the transition points of the site s:

$$\begin{gathered} t_{\text{start}}(a_{\text{sw}}(r, p_i, m_k, m_l, s)) \geq t_{\text{end}}(a_{\text{obs}}(r, p_j, s)) \\ p_i \in P_{tr} \cap P_s,\ p_j \in P_{obs} \cap P_s \end{gathered} \tag{2}$$

This dependency highlights how symbolic preconditions translate into temporal constraints.

[1] a more extensive modeling of other actions can be found in our earlier paper [De La Rochefoucauld et al., 2024]

4.1.3 Planning problem

Building on the actions models, the planning problem can now be formalized as the generation of an executable plan $\mathscr{P}$ such that:

- Ensures all sites are observed, sampled, and data is communicated;
- Satisfies coalition requirements (e.g., concurrent sampling and relay when required);
- Respects the robots' kinematic constraints, medium-specific motion models, and transition points;
- Minimizes the total mission makespan, while fulfilling all task and team constraints.

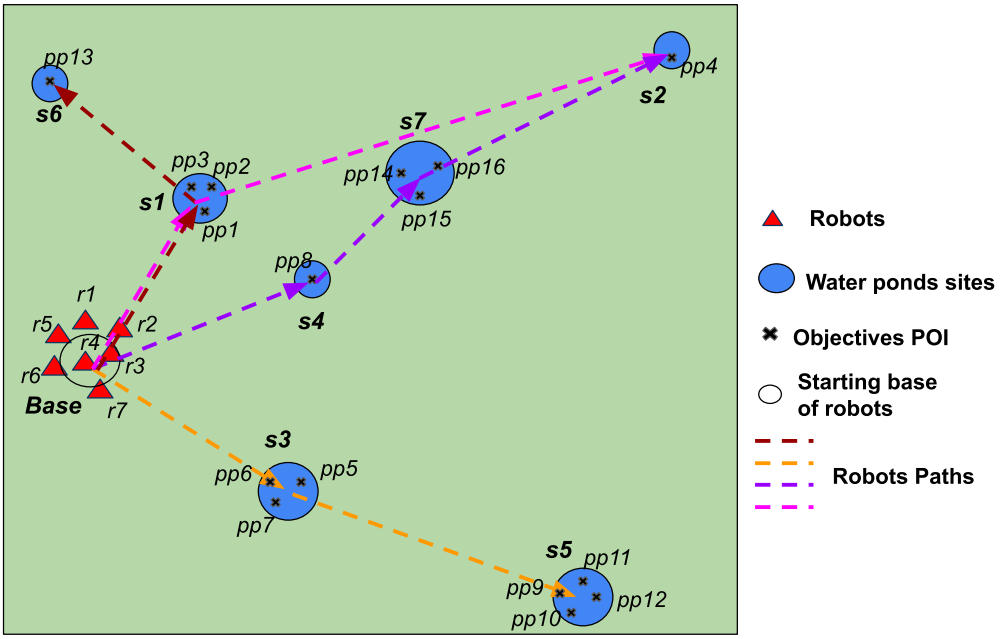


Figure 2: Representation of a multi-medium, multi-robot mission for water-pollution sampling across multiple bodies of water. Sampling points P_{spl} (denoted pp).

Because the MRTA variant grows combinatorially even at moderate scales, we decompose the global instance with AMA-PLAN into modular subproblems that bound branching and surface temporal structure for execution.

4.2 Planning with AMA-PLAN: Modular Mission Decomposition

This subsection recalls the AMA-PLAN planning framework originally introduced in [De La Rochefoucauld et al., 2024]. AMA-PLAN was designed to decompose large-scale multi-robot missions,

especially in complex trans-media environments, into smaller symbolic problems that can be individually solved by classical PDDL planners. It does not target theoretical optimality, but prioritizes good solutions within execution-time budgets for complex instances that are otherwise impractical for general-purpose planners. The core contribution of AMA-PLAN is the restructuring of the mission planning pipeline into three distinct stages that exploit domain knowledge to improve scalability and structure:

1. **(Stage 1) Spatio–cost clustering**: Sites are grouped into clusters based on spatial proximity and task cost heuristics (intra-site Traveling Salesman Problem (TSP)), using a weighted k-means approach. Each cluster is used to separate sites for greedy allocation. The order of sites is readjusted for optimal paths.

2. **(Stage 2) Robot Allocation with Resolution Cost Estimation**: A cost-aware greedy algorithm explores alternative site–robot assignments by minimizing inter-site travel and estimating per-site resolution costs (e.g., sampling, relay, navigation). We prune the search space by retaining only near-optimal coalitions, which reduces the enumeration load while preserving diversity. This stage is the main optimization bottleneck.

3. **(Stage 3) Paths assignment and path sequencing**: Based on the best allocation scenario from Stage 2, robots are then assigned concrete paths. Sites are ordered into symbolic subproblems, and inter-path dependencies are extracted to guide execution coordination. Teams emerge as coalitions of robots whose paths converge on shared sites.

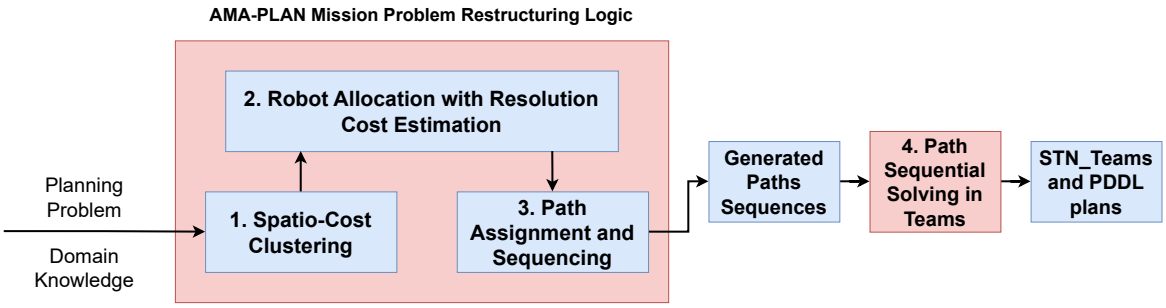
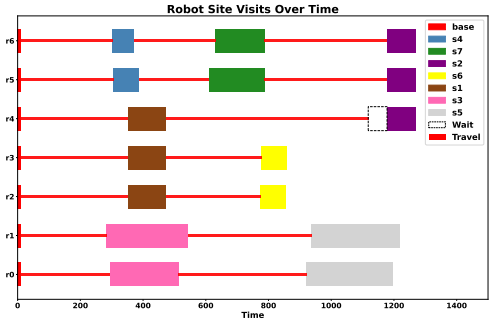


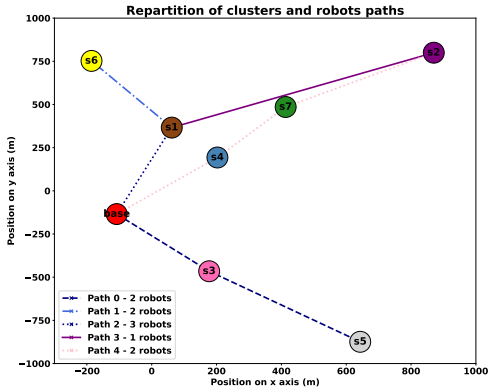
Figure 3: AMA-PLAN decomposes the MRTA planning problem into clusters of sites, assigns paths of sites to robots forming coalitions in teams, and produces sequentially linked subplans.

Figure 3 summarizes the AMA-PLAN pipeline: the global MRTA problem is partitioned into site clusters with robot allocations per cluster; resulting paths are then resolved into symbolic subplans via a sequence of PDDL solver calls.

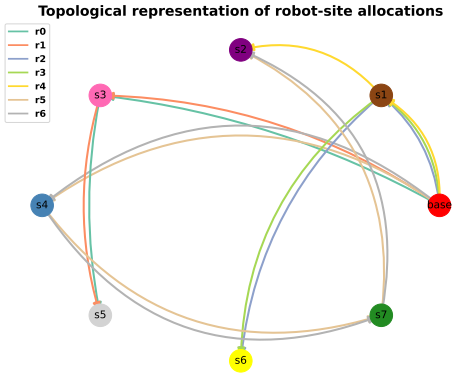
To illustrate the outcome of the AMA-PLAN allocation and resolution process, we present a mission scenario involving 7 robots and 7 sites, representative of our typical multi-robot deployments. This scenario showcases how a symbolic planning problem is decomposed into team-based subproblems with assigned paths of sites and scheduled resolution.



(a) Execution timeline showing site visits and wait durations due to team switching



(b) Cluster decomposition and robot paths, grouped by teams.



(c) Topological dependency graph of robot-site visits.

Figure 4: AMA-PLAN decomposition and allocation result for a mission involving 7 robots and 7 sites including the base.

As illustrated in Figure 4, the AMA-PLAN output is presented in three dimensions. First, the execution timeline in Figure 4a reflects the sequencing of site visits per robot, including planned idle durations that arise when a robot temporarily switches teams, for example, to support complex operations at another site.

In Figure 4b, the geographical proximity of sites is reflected in the formation of teams responsible for visiting these locations. Robots allocated to a common path traverse a shared trajectory across the sites within a given cluster, a path being determined by both spatial proximity, task requirements, and the cost estimation of each site. Finally, Figure 4c presents a topological graph of robot-to-site transitions. It helps emphasize the shared trajectories of robots and the dynamic reallocation of robot 4 with color-coded transitions by robots.

The collective analysis of these figures underscores the adaptability and organizational framework inherent within the AMA-PLAN planning layer. Robots can dynamically change teams across clusters based on role requirements and action cost estimation, leading to improved re-

source utilization and mission efficiency. This modular abstraction is essential for the downstream integration of temporal reasoning, thereby enabling the STN-based execution engine to supervise synchronization and address delays without compromising the integrity of the mission plan.

The output comprises temporally annotated, team-specific symbolic subplans with explicit interdependencies. This representation directly supports the execution system AMA-EXEC, enabling scalable plan execution across distributed robot teams.

Despite producing feasible subplans, the initial AMA-PLAN lacked a formal mechanism to manage global temporal dependencies and synchronization constraints between distributed teams. Symbolic paths are resolved sequentially (Figure 5) and remain locally valid, but consistent coordination across teams—especially when they share sites—requires an explicit temporal layer.

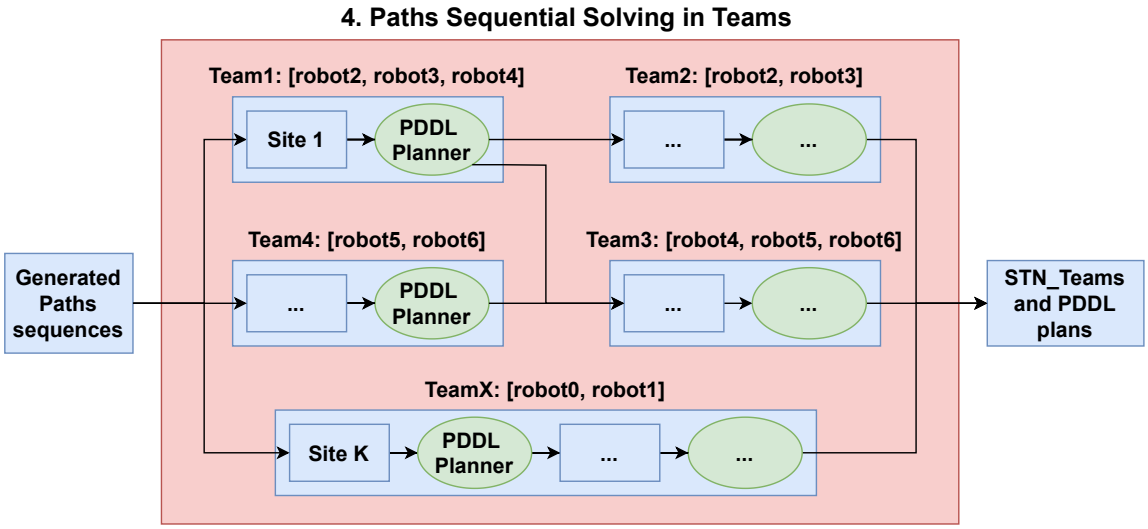


Figure 5: AMA-PLAN subproblems are sequentially resolved and merged into a global solution of constrained subplans in $\text{STN}_{\text{teams}}$.

These limitations motivated the next section, where we formally extend AMA-PLAN with the $\text{STN}_{\text{teams}}$ constructed directly from the distributed PDDL subplans. This extension facilitates inter-paths synchronization and consistency during execution, thereby bridging the gap between symbolic planning and distributed temporal reasoning.

4.3 AMA-PLAN Enhancements for Execution-Ready Plans

While the original AMA-PLAN pipeline successfully decomposes the global MRTA problem into robot-specific site sequences and resolves each site as an individual subproblem, it does not account for global timing consistency or inter-team coordination improvements. When multiple teams converge on shared sites at different times, it results in higher idle delays and long

idle plan execution. To address these limitations and enable distributed execution and runtime supervision, AMA-PLAN was extended with a fourth stage (stage 4) to restructure the planning process around team-based symbolic subplans and to introduce temporal reasoning through $\text{STN}_{\text{teams}}$, enabling synchronized, execution-ready outputs.

After site clustering and robot path assignment, we introduce the following execution-oriented solving structure:

4.3.1 Team Abstraction and Canonical Sequential-Link Generation

As in the original AMA-PLAN pipeline, robots are grouped into teams based on identical site assignments. However, each team now retains sequential connections to its predecessors and successors, and is annotated with its robots, assigned sites, initial state, and final state.

4.3.2 Temporal $\text{STN}_{\text{teams}}$ Construction over Paths.

The high-level $\text{STN}_{\text{teams}}$ is built over these sequential connections. Each node represents a symbolic plan assigned to a team, and edges encode precedence constraints due to shared robot participation or resource dependencies. Let $\mathscr{T} = \{\text{team}_1, \ldots, \text{team}_k\}$ be the set of symbolic team plans. The $\text{STN}_{\text{teams}}$ $\mathscr{G}_{\text{teams}} = (V, E)$ is then defined as:

$$V = \mathscr{T},$$
$$E = \{(\text{team}_i, \text{team}_j) \mid \text{team}_i \prec \text{team}_j\}$$

where $\text{team}_i \prec \text{team}_j$ denotes a symbolic precedence constraint induced by various dependencies like shared robot participation and temporal constraints.

4.3.3 Synchronization Node Insertion.

When a team has multiple predecessors and requires robot coordination at the first site, synchronization nodes are evaluated. For each team $team_i$, the system:

1. Computes earliest arrival times for all robots from predecessor teams.
2. Evaluates all subsets of predecessors that could participate in a sync.
3. Computes idle time for each sync scenario and compares it to a no-sync baseline.
4. Inserts one or more synchronization nodes if idle time is reduced.

This mechanism enforces robot arrival alignment only when beneficial. The process is summarized in Algorithm 1 and Figures 6 and 7 show the comparison between pre-synchronization and unsolved $\text{STN}_{\text{teams}}$ graph $\mathscr{G}_{\text{teams}}$ and the execution-ready $\text{STN}_{\text{teams}}$ graph $\mathscr{G}'_{\text{teams}}$.

Algorithm 1 minimizes idle time between teams and enforces joint or closely aligned robot arrivals when beneficial, ensuring tighter team coordination at transition sites.

Algorithm 1 Insert Synchronization Nodes for Coordinated Execution.

Input: Execution teams $\mathscr{T} = \{\text{team}_1, \ldots, \text{team}_k\}$, $\text{STN}_{\text{teams}}$ graph $\mathscr{G}_{\text{teams}} = (V, E)$, travel times T_{travel}
Output: Augmented $\text{STN}_{\text{teams}}$ $\mathscr{G}'_{\text{teams}}$ with synchronization nodes
for all teams $team_i \in \mathscr{T}$ with multiple predecessors **do**
    $R_i \leftarrow$ assigned robots to $team_i$
    $T_{\text{arr}} \leftarrow$ estimated arrival times for each $r \in R_i$
    $T_{\text{idle}}^{\text{base}} \leftarrow \max T_{\text{arr}} - \min T_{\text{arr}}$
    $T_{\text{best}} \leftarrow T_{\text{idle}}^{\text{base}}$
    $C_{\text{best}} \leftarrow \emptyset$
    for all subsets $C \subseteq \text{Predecessors}(team_i)$ **do**
        $T_{\text{arr}}^{C} \leftarrow$ recomputed arrival times with sync delay from C
        $\Delta T \leftarrow \max T_{\text{arr}}^{C} - \min T_{\text{arr}}^{C}$
        if $\Delta T < T_{\text{best}}$ **then**
            $T_{\text{best}} \leftarrow \Delta T$
            $C_{\text{best}} \leftarrow C$
        end if
    end for
    if $C_{\text{best}} \neq \emptyset$ **then**
        for all $team_j \in C_{\text{best}}$ **do**
            Create sync node $sync_{i,j}$ for shared robots
            $V \leftarrow V \cup \{sync_{i,j}\}$
            $E \leftarrow (E \setminus \{(team_j, team_i)\}) \cup \{(team_j, sync_{i,j}), (sync_{i,j}, team_i)\}$
        end for
    end if
end for
return Augmented $\text{STN}_{\text{teams}}$ $\mathscr{G}'_{\text{teams}} = (V, E)$

4.3.4 Execution-Time Estimation and Path Splitting.

Symbolic execution teams consist of sequences of sites and are incrementally transformed into a chain of subproblems. Each subproblem corresponds to a site (or a small set of sites) and is generated using the robots' updated states resulting from their previous subproblem's plans, ensuring continuity of execution context across the $\text{STN}_{\text{teams}}$. Subproblems are solved using a PDDL planner (we tested several via the UP [Micheli et al., 2025], but we typically use OPTIC [Benton et al., 2012]), and the resulting plans are annotated with their estimated durations. If a plan's cumulative duration or action count exceeds a predefined threshold, the path is automatically split into synchronized subteams that cover all assigned sites. This improves execution tractability and localizes the impact of failures.

4.3.5 Subproblem Generation and Validation Structure.

Each execution team is converted into a directory of symbolic subproblems, domain files, plans, and merged solutions. These files are used for both execution and post-failure revalidation (explained in Section 5). The planning sequence ensures continuity of robot states across subproblems.

4.3.6 Final $\text{STN}_{\text{teams}}$ Graph and Execution-Ready Output.

After subproblem generation and validation, we obtain the final $\text{STN}_{\text{teams}}$, which serves as the temporal backbone of the execution schedule. It includes:

- Nodes: team paths, synchronization nodes,
- Edges: sequential constraints with minimum gaps,
- Metadata: estimated execution times, robot participants, site locations.

This structure is provided to the STN-CONTROLLER, which monitors timing, handles drift, and enforces synchronization and dependency constraints at runtime. The plans are therefore *execution-ready*: they are temporally annotated, robot states are consistent across subproblems, and the $\text{STN}_{\text{teams}}$ embeds the constraints required for direct handoff to execution.

4.4 Overall complexity of AMA-PLAN

Keeping only dominant terms across stages, the overall complexity is

$$T_{AMA-Plan} = \mathcal{O}\big(|\mathscr{S}|^2\, 2^{|\mathscr{S}|/2}\big) + \mathcal{O}\big(|\mathscr{S}|\,|\mathscr{R}|^2\, \mathscr{B}^{|\mathscr{S}|/2}\big) + \sum_i T_{\text{optic}}(L_i). \tag{3}$$

where $\mathscr{S}$ denotes the set of non-base sites, $\mathscr{R}$ the set of robots (with cardinalities $|\mathscr{S}|$ and $|\mathscr{R}|$), $\mathscr{B}$ is the effective post-pruning per-site branching factor induced by Stage 2 (*Allocation resolution cost estimation*)—i.e., the size of the small retained set of near-optimal coalitions per site

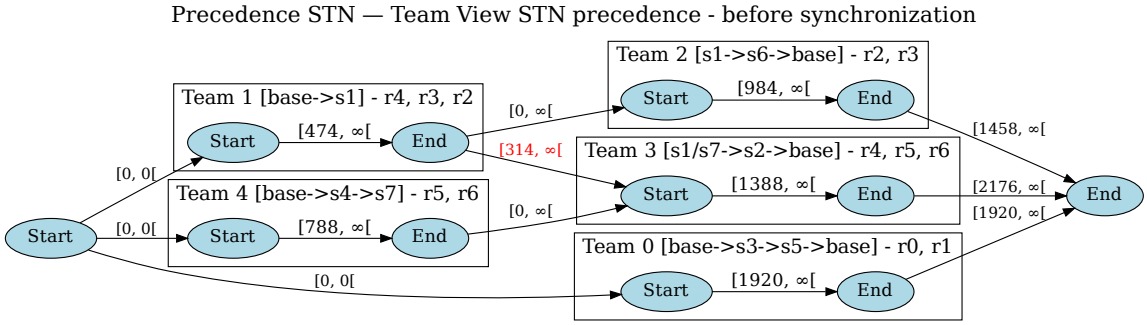


Figure 6: Initial $\text{STN}_{\text{teams}}$ structure, with no synchronization adjustment.

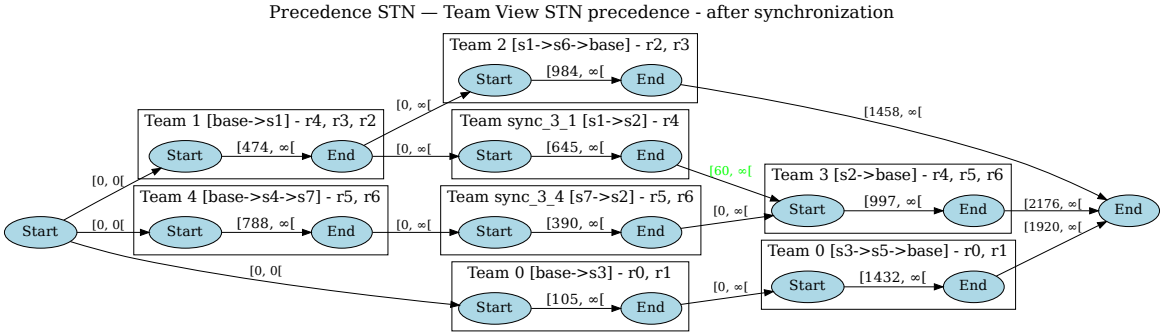


Figure 7: Improved $\text{STN}_{\text{teams}}$ with synchronized path nodes and durations from PDDL plans.

(data-dependent; worst case equals the unpruned option count), and L_i the size of early-split subproblems for OPTIC. The first term comes from exact intra-cluster TSP ordering during spatio–cost clustering; the second from pruned cross-site composition with greedy cost estimation; the last term is the sum of OPTIC solver times over bounded subproblems produced by $\text{STN}_{\text{teams}}$ synchronization and early-split. All remaining components (fixed-iteration k-means, weight construction, greedy picks, STN updates) are lower-order in our implementation.

4.4.1 Scope and limitations.

Because Stage 1 uses weighted Euclidean k-means and Stage 2 employs greedy allocation over a pruned band, AMA-PLAN is heuristic: elongated/anisotropic layouts, severe per-site workload imbalance or interleaved clusters can degrade solution quality or enlarge $\mathscr{B}$. We mitigate these effects with travel-time weighting and multiple restarts (Stage 1), band-pruning (Stage 2, controlling $\mathscr{B}$), and early-split caps in $\text{STN}_{\text{teams}}$.

4.4.2 Bottleneck implication.

As $|\mathscr{S}|$ grows, the scenario-construction/assignment term $\mathscr{O}(|\mathscr{S}|\,|\mathscr{R}|^2\,\mathscr{B}^{|\mathscr{S}|/2})$ becomes dominant; increasing $|\mathscr{R}|$ at fixed $|\mathscr{S}|$ primarily inflates the $|\mathscr{R}|^2$ factor, while STN consistency checks remain lightweight.

The next section presents how AMA-EXEC monitors the execution-ready plans produced by AMA-PLAN and reacts to temporal drift, action delays, and failures at runtime.

5 Temporal Reasoning and Failure-Resilient Execution

This section presents the mathematical modeling, temporal reasoning, and failure-resilient mechanisms and algorithms integrated in AMA-EXEC. Building on the symbolic plans and synchronized structures produced by AMA-PLAN, AMA-EXEC enables robust real-time execution in uncertain environments. It introduces runtime models that monitor temporal constraints, handle drifts and delays, and analyze failures in order to identify the parts of the $\text{STN}_{\text{teams}}$ plan that are impacted.

5.1 Execution-Time Challenges in Real-World Deployment

Symbolic plans generated by AMA-PLAN are typically temporally annotated and logically coherent. However, they rely on idealized assumptions: perfectly synchronized execution across agents, fixed action durations, and failure-free environments. These assumptions rarely hold in practice, especially in trans-media, multi-robot scenarios that involve complex vehicle capabilities and behaviors, medium transitions, and partially observable conditions.

During execution, agents may experience delays, asynchronous progress, or misaligned temporal dependencies, particularly when they are operating in separate teams. For instance, a sampling action may begin prematurely before a prerequisite data-translation action has completed, producing invalid behavior. Furthermore, failures such as robot crashes or the inability to reach a site can impact mission feasibility in localized ways, often without invalidating the entire mission.

As discussed before, PlanSys2's traditional BT-based execution engine and ROS 2-based planning frameworks do not monitor temporal coherence at runtime, nor do they provide fine-grained control over how failures are diagnosed or mitigated. As a result, minor execution drifts or localized robot failures may lead to over-conservative responses such as full replanning or mission failure.

AMA-EXEC addresses these limitations through a dual-layered execution framework. The STN-CONTROLLER performs continuous supervision of action timing and dependencies, ensuring that symbolic-temporal constraints are upheld during distributed execution. In parallel, the DEM handles failure detection and recovery, including constraint-aware reallocation and replanning decisions. Together, these modules enable robust and adaptive execution across multi-agent trans-media teams.

To address these challenges, AMA-EXEC relies on two supervisory modules: the STN-CONTROLLER, which maintains temporal coherence and detects drift, and the DEM, which diagnoses failures and coordinates recovery.

5.2 Temporally Aware Distributed Execution: The STN-CONTROLLER

The STN-CONTROLLER is the first supervisory module of AMA-EXEC. It supervises distributed execution by interpreting symbolic plans as temporal dependency graphs, tracking their live progression, and enforcing synchronization when necessary. Before describing its execution-time monitoring loop, we first introduce the formal temporal model on which it relies.

5.2.1 Formal Temporal Model and Dependency Graph Structure

The STN-CONTROLLER executes symbolic plans from the $\text{STN}_{\text{teams}}$ using a multi-layered dependency graph. This extended model, referred to as STN_{act}, captures both the temporal structure and semantic dependencies of individual robot actions during execution. This structure is used by the STN-CONTROLLER to supervise runtime execution, ensure correct action monitoring and enforce synchronization constraints.

Temporal Annotation We define a symbolic plan $\mathscr{P}$ as a set of actions $\{a_1, \ldots, a_n\}$, where each action a_i is annotated with:

- s_i: planned start time,
- d_i: expected duration,
- $e_i = s_i + d_i$: planned end time.

The core temporal structure of STN_{act} is defined as:

$$\mathscr{G}_{\text{act}} = (V, E), \quad V = \{s_i, e_i\}, \quad E = \{(e_i + \varepsilon \leq s_j)\}$$

where ε is a buffer margin to absorb uncertainty or enforce loose synchronization.

To better capture execution semantics and mission logic, we extend this structure into a multi-layered dependency graph derived from the PDDL plan and domain constraints file:

$$\mathscr{G}_{\text{act}} = (V, E_T, E_C, E_R) \tag{4}$$

where:

- E_T: *Sequential (temporal) dependencies* Represents strict ordering between actions executed by the same robot (e.g., navigation must finish before sampling), (this is similar to the leads_to relation in Allen logic [Allen, 1981]),

- E_C: *Concurrency constraints* Links actions across robots that must be executed concurrently or with minimal offset (e.g., `sample` must be contained_by (in Allen logic) a `translate_data` action),

- E_R: *Repeated action equivalences* Denotes semantically equivalent actions (with same parameters) or refer to similar operations, often used to group and track similar actions across time,

Illustrative Example.

We consider the following sequence from a plan involving two robots visiting a site, where 'robot0' `translate_data` (for relays) and 'robot1' performs `sample`, both actions being constrained such that a `sample`(a_{spl}) is always contained in another robot's `translate_data` (a_{trd}) action on the same site s:

$$\begin{aligned} \forall r_u, r_v \in \mathscr{R},\ \forall p_i \in P_{spl} \cap P_s,\ \forall p_j \in P_{tr} \cap P_s, \\ m_{r_u} = m_{r_v} = \text{aquatic}, \\ t_{\text{start}}(a_{\text{spl}}(r_u, p_i, s)) \geq t_{\text{start}}(a_{\text{trd}}(r_v, p_j, s)) \\ \wedge\ t_{\text{end}}(a_{\text{spl}}(r_u, p_i, s)) \leq t_{\text{end}}(a_{\text{trd}}(r_v, p_j, s)) \end{aligned} \tag{5}$$

The following action sequence (with symbolic temporal annotations) illustrates our extended STN_{act} dependency structure from the PDDL plan:

- a_1: `(translate_data robot0 sp9 site3 A)` $s_1 = 334, d_1 = 45, e_1 = 379$

- a_2: `(sample robot1 pp7 site3)`
  $s_2 = 349, d_2 = 30, e_2 = 379$

- a_3: `(navigation_water robot1 pp7 pp6 site3)`
  $s_3 = 379, d_3 = 17.9, e_3 = 397$

- a_4: `(translate_data robot0 sp9 site3 B)` $s_4 = 382, d_4 = 45, e_4 = 427$

- a_5: `(sample robot1 pp6 site3)`
  $s_5 = 397, d_5 = 30, e_5 = 427$

From this, the dependency graph $\mathscr{G}_{\text{act}} = (V, E_T, E_C, E_R)$ includes:

- **Sequential dependencies (E_T)**
  - $(a_1, a_4) \in E_T$, robot0 sequence
  - $(a_2, a_3) \in E_T, (a_3, a_5) \in E_T$, robot1 sequence

- **Concurrency constraints (E_C)**
  - $(a_1, a_2) \in E_C$, first concurrent actions

- $(a_4, a_5) \in E_C$, second concurrent actions

- **Repeated action Constraint (E_R)**
  - $(a_1, a_4) \in E_R$, e.g: same `translate_data` action parameters but with different `sample` action related at a different time. An index representing their order (A $\rightarrow$ Z,) is added to the action automatically when a constraint is added. This can be seen in the example above with `translate_data robot0 sp9 site3 A` and `translate_data robot0 sp9 site3 B`.

Algorithm 2 Extraction of Multi-Layered Dependencies from a Plan

Input: Symbolic plan $\mathscr{P} = \{a_1, \ldots, a_n\}$ with timing and parameters
Output: $\mathscr{G}_{\text{act}} = (V, E_T, E_C, E_R)$

$V \leftarrow \mathscr{P}$ {Actions as nodes}
$E_T, E_C, E_R \leftarrow \emptyset$
for each pair $(a_i, a_j) \in \mathscr{P}^2$, with $i \neq j$ **do**
  if a_i and a_j share a robot **and** a_j starts after a_i within tolerance **then**
    $E_T \leftarrow E_T \cup \{(a_i, a_j)\}$ {Temporal order}
  else if a_i and a_j operate on same site or PoI **and** roles are complementary **then**
    $E_C \leftarrow E_C \cup \{(a_i, a_j)\}$ {Coordinated concurrency}
  else if a_j is a repeated instance of a_i (e.g., repeated `translate_data` on same site) **then**
    $E_R \leftarrow E_R \cup \{(a_i, a_j)\}$ {Semantic repetition}
  end if
end for
return $\mathscr{G}_{\text{act}} = (V, E_T, E_C, E_R)$

Algorithm 2 details the construction of the multi-layered dependency graph $\mathscr{G}_{\text{act}}$ from the symbolic plan $\mathscr{P}$. It examines inter-action relationships using semantic information, temporal annotations, and shared roles or PoI involvement as shown in Figure 8. This structure serves as the basis for execution-time synchronization and enforcement of constraints in the STN-CONTROLLER.

This layered structure allows mission controllers to track execution with well established semantics, enabling constraint-based adaptation, delay diagnosis, and real-time synchronization enforcement.

Building on this formal model, we now turn to the execution-time architecture of the STN-CONTROLLER, describing how it interfaces with the rest of the system and supervises distributed teams in practice.

5.2.2 Architecture and Execution-Time Monitoring

The STN-CONTROLLER is the centralized supervision node responsible for maintaining the temporal consistency of distributed symbolic plans executed in parallel by robot teams. It interfaces with both the TLCM and per-robot BT executors, and serves five key purposes:

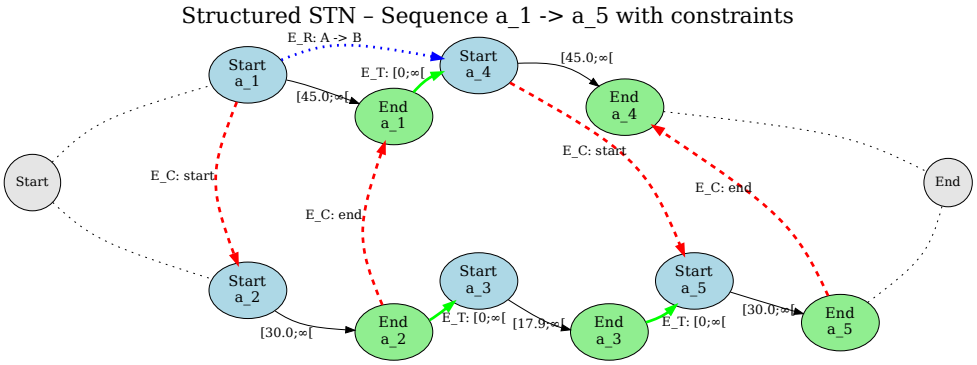


Figure 8: Multi-layered STN_{act} dependency graph derived from symbolic plan. Edges represent: Temporal dependencies (green line), concurrency constraints (red dashed line), and redundancy (blue dotted line) constraints between annotated start and end nodes.

- Creating the STN_{act} by building dependency graphs $\mathscr{G}_{\text{act}}$ from the PDDL plans.
- Maintaining a dedicated STN_{act} instance and action tracking table for each active team.
- Continuously monitoring real-time feedback from execution to assess a robot's action and plan progression in its team's STN_{act} .
- Enforcing the symbolic and temporal constraints as introduced in STN_{act}: temporal (E_T), concurrency (E_C), and semantic equivalence (E_R).
- Propagating delay commands to robots and logging failure contexts if violations are detected.

Each team is supervised through an independent timer loop evaluating constraint satisfaction during execution using up-to-date action states published by each robot. Internally, each STN_{act} instance includes expected durations between start and end nodes for actions, as well as inter-action dependencies derived from plan parsing.

During execution, if a violation of temporal constraints is detected, such as an action finishing before its dependent peer (e.g., `translate_data` ending before `sample`) the controller reacts by computing and issuing a delay signal. The affected robot is instructed to extend execution of the ongoing action before sending success to its BT executor (typically representing a wait) to realign the team's plan temporally. This mechanism ensures robust and synchronized execution when dealing with execution delays and drift.

Algorithm 3 illustrates the containment-based synchronization process used by the STN-CONTROLLER to enforce concurrency constraints (E_C) at execution time. For each team, the controller iterates over active action pairs and monitors whether the upcoming E_C constraints of the team are still satisfied. It introduces delays to the appropriate robots to realign their execution and satisfy containment of the sampling actions when needed.

Algorithm 3 Concurrency Constraints (E_C) Synchronization in the STN-CONTROLLER

Input: Team T, current actions $\mathscr{A}_T$ with expected start/end times $(\hat{s}_i, \hat{e}_i)$
while team T is active **do**
  for nexts $(a_i, a_j) \in E_C$ **do**
    if a_j not started **then**
      if $\hat{s}_j < \hat{s}_i$ **then**
        Delay a_j to start after a_i
      end if
      if $\hat{s}_j + d_j > \hat{e}_i$ **then**
        Delay a_i to extend containment
      end if
    else if a_j running **and** $\hat{e}_j > \hat{e}_i$ **then**
      Delay a_i to preserve containment
    end if
  end for
end while

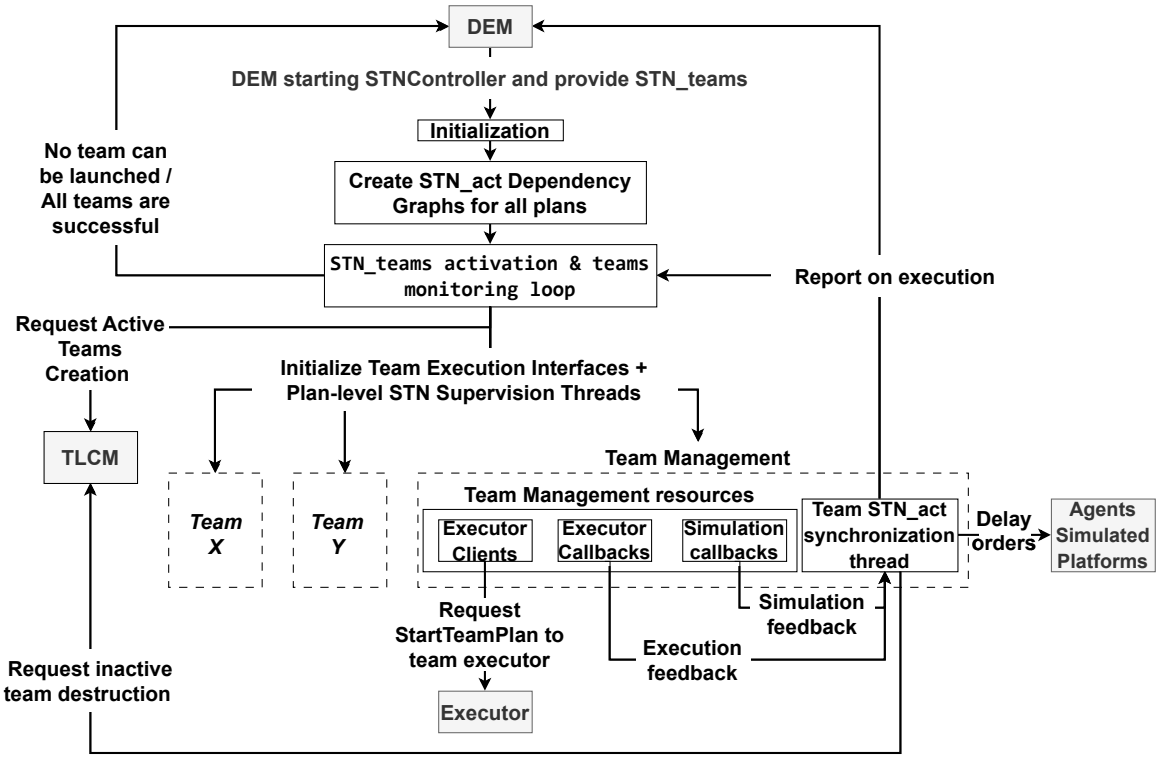


Figure 9: STN-CONTROLLER execution architecture. Teams from $\text{STN}_{\text{teams}}$ are supervised in parallel via dedicated synchronization threads informed by STN_{act}.

Figure 9 illustrates the expanded runtime execution architecture of the STN-CONTROLLER. After initialization by the DEM, it prepares STN_{act} graphs for all teams of the $\text{STN}_{\text{teams}}$ and launches a centralized team monitoring loop. This loop activates robot teams based on readiness and assigns each one a dedicated supervision thread.

Each thread interfaces with its team's action executor and robots simulations, continuously comparing live feedback with STN_{act} constraints. As seen in Algorithm 3, this allows the STN-CONTROLLER to maintain alignment of actions and execution without resorting to replanning.

Teams' lifecycles (creation and removal) are coordinated with the TLCM, and updated execution statuses are reported upstream to the DEM for potential diagnostic or replanning decisions in case of failure. This modular structure allows for plan-level temporal supervision, enabling scalable and resilient execution of distributed multi-robot missions.

The modularity of this design allows for parallel monitoring of multiple teams, each with its own plan, STN_{act} instance, and feedback loop. This structure scales to heterogeneous execution speeds, inter-robot variability, and enables smooth transitions across symbolic-temporal plans.

Beyond temporal supervision and delay injection, AMA-EXEC must also handle execution failures in a structured manner. To this end, the STN-CONTROLLER captures violations and compiles failure reports, which are then consumed by the DEM for diagnosis and recovery.

5.2.3 Failure-Aware Execution Reporting and Time Budget Estimation

When execution significantly violates the STN_{act} constraints or an action failure is detected, the STN-CONTROLLER logs structured failure data in $\mathscr{F}$ and performs a forward estimation of the remaining duration of the failing team's plan, along with those of others currently executing teams. This estimation is performed using the current state of the STN_{act} dependency graph, which includes action progress, expected end times, and active constraints. The process includes:

- Identifying the failing team, robot, site, and point of interest,
- Structuring the failure context (including causal and temporal metadata),
- Estimating remaining execution time for the affected team,
- Identifying parallel teams currently executing and their estimated completion times,
- Transmitting this information to the DEM to support constraint propagation and reallocation strategies.

This mechanism allows the DEM to make informed decisions on whether to reassign idle teams, wait for execution completion, or trigger partial replanning. The failure report $\mathscr{F}$ acts as the formal bridge between runtime monitoring (STN-CONTROLLER) and strategic recovery. It provides the DEM with structured causal and temporal context, which forms the input to its diagnosis stage described next.

These supervision and delay injection mechanisms enable AMA-EXEC to maintain symbolic and temporal coherence across distributed teams, adapting to execution drift and action

misalignment without centralized blocking or unnecessary replanning. In doing so, they establish the runtime backbone for robust multi-team coordination, and hand off structured reports to the DEM for diagnosis, as described next.

5.3 Failure Diagnosis and Impact Assessment

Building on the failure report $\mathscr{F}$ issued by the STN-CONTROLLER, the DEM initiates a recovery pipeline with three stages:

1. **Failure Context Parsing:** Extracts key metadata: robot, team, point and site of failure, failed action, and timing.
2. **Propagation Across Dependencies:** Traverses the inter-team dependency graph to identify all mission segments potentially affected by the failure. This graph encodes not only causal relations but also symbolic links (e.g., shared roles or resources) and temporal constraints (from STN), enabling richer and more accurate failure impact propagation than a standard task graph.
3. **Constraint-Aware Diagnosis:** Evaluates the executability of each affected team, checking constraint satisfaction, plan validity, timing feasibility, and potential for reallocation.

Each team is independently evaluated along the following dimensions:

- **Constraint satisfaction**: minimum robot counts, role requirements, capabilities, and other domain constraints must hold at every site.
- **Plan validity**: the remaining plan is structurally executable with the updated robot capabilities. We assess this with VAL [Howey et al., 2004] in plan-continuation mode, which checks the feasibility of all remaining steps under current conditions without revalidating completed actions or full mission goals.
- **Timing feasibility**: symbolic–temporal constraints (e.g., orderings and STN bounds) remain satisfiable.
- **Reallocatability**: a reassignment of available robots can restore executability while respecting constraints.

The results are aggregated into a diagnostic map $\mathscr{D}$, which annotates each affected team with flags (invalid, constraint violation, timing violation, reallocatable). This report does not categorize failures into predefined types but instead enables the planning module (AMA-PLAN) to select the minimal replanning scope based on structural impact.

The DEM also computes a numerical *impact depth score*, representing how severely a failure compromises the original team's future actions. This provides a quantitative basis for AMA-PLAN to weigh robot exclusion or reallocation.

The next subsection describes how reallocation decisions are made from the diagnostic report and how allocation candidates are scored.

5.4 Allocation Scoring and Failure Recovery Decision

Given a diagnostic map $\mathscr{D}$, the DEM must decide whether recovery is possible through reallocation or whether partial replanning is necessary. This decision involves scoring candidate robot–team assignments and evaluating the validity of updated team plans. If all invalid teams are reallocatable, candidate robot assignments are generated and evaluated based on former path structures.

5.4.1 Per-Robot Assignment Score

Let r be a robot and $team_i$ a candidate team. The cost of assigning r to $team_i$ is given by:

$$\begin{aligned} \text{score}(r, team_i) = \text{idle}(r, team_i) + \\ \text{dist}(r, team_i) + \text{penalty}(r, team_i) \end{aligned}$$

Where:

$$\begin{aligned} \eta(r, team_i) &= t_r^{\text{avail}} + d\left(team_r^{\text{last}},\ team_{team_i}^{\text{start}}\right) \\ t_{team_i}^{\text{start}} &= \max_{r' \in R_{team_i}} \eta(r', team_i) \\ \text{idle}(r, team_i) &= \max\left(0,\ t_{team_i}^{\text{start}} - \eta(r, team_i)\right) \\ \text{dist}(r, team_i) &= \left\| team_r^{\text{last}} - team_{team_i}^{\text{start}} \right\|_2 \end{aligned}$$

- t_r^{avail}: robot r's earliest availability time,
- $d()$: travel time cost function,
- $team_r^{\text{last}}$: last known site visited by r,
- $team_{team_i}^{\text{start}}$: first site of $team_i$,
- $\mathscr{R}_{team_i}$: set of robots assigned to team $team_i$,
- $\eta(r, team_i)$: estimated arrival time of r at $team_i$.

Note that a robot's availability time (t_r^{avail}) reflects both its current state and ongoing execution obligations, ensuring that idle time accounts for realistic timing, not just whether a robot is currently idle.

5.4.2 Proportional Reassignment Penalty.

To discourage unnecessary disruption, we assign a penalty to each robot based on how much its reassignment modifies the original team and the severity of the impact:

$$\text{penalty}(r, team_i) = \begin{cases} \gamma \cdot \left(1 - \dfrac{|T_{\text{rem}}|}{|T_{\text{orig}}|}\right) \cdot \delta, & (i) \\ \gamma, & (ii) \\ 0, & (iii) \end{cases} \tag{6}$$

Here, γ is a user-defined scaling factor (set to 3.0 in our experiments) that controls the relative weight of reassignment penalties. T_{orig} is the set of robots originally assigned to the team, and T_{rem} the team after removing r. The factor $\delta = 1$ if the removal violates constraints (e.g., not enough robots remaining), and 0.5 otherwise. This penalty formulation results in three typical cases:

- case (i): r is reassigned from a team that is not executing, incurring a penalty scaled by future team disruption and constraint violation;
- case (ii): r was not initially assigned to any path (e.g., its previous path has finished), and receives a flat penalty γ;
- case (iii): r remains on its original path, resulting in no penalty.

This penalty guides the allocation toward minimal disruption of valid team distribution.

5.4.3 Valid Allocation Generation

GenerateValidAllocations (GVA) constructs candidate assignments by generating variations of the affected paths after applying the failure's effects. It considers only robots not involved in currently executing or fixed plans. For each candidate team, it verifies the absence of constraint violations such as insufficient team size, missing capabilities, or broken relay/switch requirements. Only allocations that restore feasibility are passed on for ranking.

5.4.4 Global Allocation Score

Once all candidate reallocation options are identified, each is scored using the average cost over its constituent robot-to-team assignments:

$$\text{score}_{\text{alloc}} = \frac{1}{|\mathscr{R}|} \sum_{r \in \mathscr{R}} \text{score}(r, team_{i_r}) \tag{7}$$

Where:

- $\mathscr{R}$ is the set of robots participating in the candidate allocation,

- $team_{i_r}$ is the team path to which robot r is assigned in that allocation

The candidate with the lowest average score is selected as the recovery plan in the `RankAllocs` function.

5.4.5 Recovery Algorithm

The recovery loop below builds the diagnostic map $\mathscr{D}$, identifying the state of the $\text{STN}_{\text{teams}}$ plans, and attaches a recovery allocation if applicable.

Algorithm 4 Progressive Diagnosis and Recovery Loop

Input: Failure event $\mathscr{F}$, Plan $\mathscr{P}$, Dependencies $\mathscr{G}_{\text{act}}$, Constraints $\mathscr{C}$, Robot states $\mathscr{R}$, World info W

Output: Diagnostic map $\mathscr{D}$

$\mathscr{D} \leftarrow \{\}$
$T_{\text{aff}} \leftarrow$ `PropagateFailure`$(\mathscr{F}, \mathscr{G}_{\text{act}})$
for each $team_i \in T_{\text{aff}}$ **do**
  $\mathscr{D}[team_i] \leftarrow$ {invalid: False, constraints: False, timing: False, realloc: False}
  if not `Validate`$(team_i)$ **then**
    $\mathscr{D}[team_i]$.`invalid` $\leftarrow$ True
    if not `ConstraintsOK`$(team_i, \mathscr{C})$ **then**
      $\mathscr{D}[team_i]$.`constraints` $\leftarrow$ True
    end if
    if `ValidReallocationExists`$(team_i, \mathscr{R}, \text{W}, \mathscr{C})$ **then**
      $\mathscr{D}[team_i]$.`realloc` $\leftarrow$ True
    end if
    if `TimingDesync`$(team_i, \mathscr{F})$ **then**
      $\mathscr{D}[team_i]$.`timing` $\leftarrow$ True
    end if
  end if
end for
if `AllValidOrReallocatable`$(\mathscr{D})$ **then**
  return $\mathscr{D}$
end if
$A \leftarrow$ `GVA`$(\mathscr{D}, \mathscr{R}, \mathscr{C}, W)$
if $A = \emptyset$ **then**
  $\mathscr{D}$.`replan_required` $\leftarrow$ True
else
  $a^* \leftarrow$ `RankAllocs`$(A, W, \mathscr{R})$
  $\mathscr{D}$.`best_allocation` $\leftarrow a^*$
end if
return $\mathscr{D}$

This model ensures a clear separation between failure interpretation and recovery decision-making. By generating a complete diagnostic map $\mathscr{D}$ that classifies each affected teams, it ensures that all possible reallocation opportunities are considered before any corrective action is triggered. When reallocation is feasible, the new optimal assignment is included directly in the report, avoiding unnecessary replanning in AMA-PLAN. This structured approach improves robustness, supports explainability, and ensures consistent precise handling of both isolated failures and cascading execution disruptions.

5.5 Execution overhead of AMA-EXEC

Execution supervision is event-driven and per-team. The STN-CONTROLLER runs at 1 Hz and performs guarded pairwise checks over the team's concurrently tracked actions, giving a worst-case $\mathscr{O}(\mathscr{A}_T^2)$ work per tick (where $\mathscr{A}_T$ is the number of actions currently executing in a team and is small in practice). On failures, DEM recovery is local: we propagate on the plan/action dependency graph in $\mathscr{O}(N+E)$, validate only affected subproblems with VAL, and, if needed, backtrack reallocation teams over the few dependent paths. These overheads are consistently dominated by AMA-PLAN costs and any additional OPTIC calls when DEM triggers replanning. In contrast to these planning costs, STN-CONTROLLER consistency checks and DEM propagation remain lightweight; observed timeouts at larger scales stem from scenario construction/assignment and occasional OPTIC calls—consistent with the $\mathscr{O}(|\mathscr{S}|\,|\mathscr{R}|^2\,\mathscr{B}^{|\mathscr{S}|/2})$ term above.

6 Evaluation

In order to evaluate the effectiveness of AMA[2] in real-time planning, execution monitoring, fault tolerance, and scalability, we conduct a comprehensive evaluation combining both illustrative case studies and large-scale statistical analyses. Our evaluation focuses on three core aspects: 1) the responsiveness of the AMA-EXEC STN-CONTROLLER temporal supervision mechanism to execution drift and PlanSys2 BT-sequential execution temporal violations; 2) the planning scalability and runtime overhead of the AMA-PLAN decomposition pipeline; and 3) the adaptable recovery behavior of the DEM in AMA-EXEC under different failure types and injection strategies. The simulator exposes the same ROS 2 actions/topics/services as our target robots (PlanSys2-compatible) and uses timing/capability parameters derived from platform specifications and field observations.

6.1 Illustrative Execution Scenarios

To demonstrate the runtime supervision and recovery capabilities of our architecture, we present four illustrative execution scenarios. The first scenario highlights how the STN-CONTROLLER ensures symbolic-temporal consistency on the STN_{act} through delay injection. The following

[2]The code is available at: https://github.com/ViDLR/ROS2PLAN_ws.

three scenarios cover failure handling via the DEM, with increasing severity and recovery cost: local reallocation, multi-path recovery, and full replanning.

6.1.1 STN_{act}-Only Delay Synchronization

During multi-robot execution, minor temporal desynchronizations may arise due to communication delays, BT scheduling inertia, or slight execution drift. In the example shown in Figure 10, a symbolic-temporal constraint (E_C) requires that the `sample` action completes before the corresponding `translate_data` action ends.

The STN-CONTROLLER continuously monitors action progress against the STN_{act}. When it detects a potential constraint violation, such as `translate_data` ending prematurely while `sample` is still executing—it dynamically injects a runtime delay into the ongoing action.

This injected delay (highlighted as orange segments with dashed outlines in the top two timelines of Figure 10) allows the symbolic order to be preserved without halting the plan or raising a failure. The plan proceeds nominally after this adjustment, and the DEM is not involved in this case.

Comparison with Existing BT Execution: In the default PlanSys2 implementation, plans are executed as strictly sequential BTs, without enforcing temporal or symbolic constraints during execution. Although a recent preprint has proposed using STN structures to build BTs with temporal flexibility [Zapf et al., 2024], to our knowledge this approach is not yet fully integrated into PlanSys2 or publicly released. Furthermore, that technique builds temporally constrained BTs, and while it addresses plan-inherent timing, it does not supervise runtime drift and non-planned variations. In contrast, our STN-CONTROLLER supervises runtime execution and enforces temporal constraints dynamically, preserving inter-action dependencies across multiple robots.

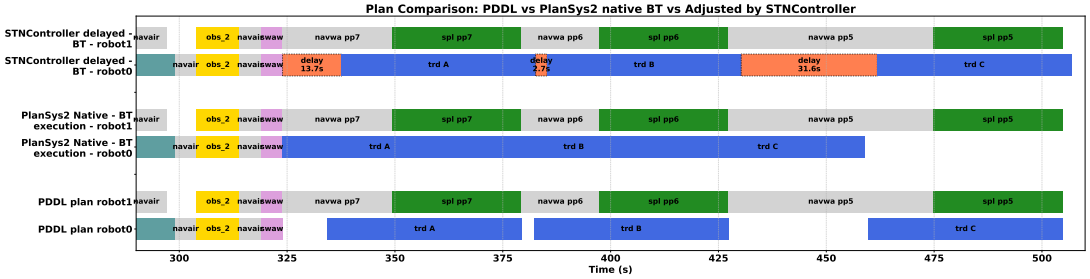


Figure 10: STN-only synchronization scenario. The top two timelines show how the STN-CONTROLLER injects runtime delays (in seconds) into the `translate_data` actions (orange dashed boxes) to respect symbolic-temporal constraints with `sample` actions, without invoking the DEM. Comparisons with PlanSys2 BT and PDDL execution timelines are shown on the four other timelines.

Observed overhead vs. PDDL baseline. On our illustrative $7|\mathscr{S}|\times 7|\mathscr{R}|$ mission (Figure 2), the STN-supervised execution finishes approximately *60–70 s* after the nominal PDDL schedule (see Figure 10 example). Most of this gap stems from the PlanSys2 PDDL→BT transformation when creating new teams and dispatch of large plans, combined with ROS 2 scheduling/QoS jitter (asynchronous BT ticks ≈ 0.2 s in multi-robot teams). The STN-CONTROLLER contributes only the additional waits strictly required to enforce inter-action temporal relations (e.g., the `sample–translate_data` containment) with buffer times. This overhead is an intended trade-off: AMA-EXEC prioritizes executability and temporal soundness under runtime uncertainties over strict adherence to the off-line schedule.

6.1.2 DEM Case 1: Localized Failure with recovery of a Single Team

Considering the plan presented in Figure 7 we introduce several levels of failure and observe how different failures will change the impact on the plan and its recovery mechanism, each case leads to a different recovery mode, which can be interpreted as local, re-allocatable and full replanning.

Impact Score Computation

To provide a consistent measure of how failures affect mission viability, we compute a S_{global} score for each recovery case. This score captures symbolic degradation, execution disruption, and structural propagation across the plan. It is defined as:

$$S_{\text{global}} = k_1 \cdot S_{\text{val}} + k_2 \cdot S_{\text{prop}} + k_3 \cdot S_{\text{realloc}} \tag{8}$$

S_{val} corresponds to the ratio of remaining $\text{STN}_{\text{teams}}$ elements that are invalidated due to the failure, as assessed by the VAL validator. The value of k_1 was set to 0.5 to represent the increased overall replanning time that results from additional unvalidated plans. The S_{prop} score is a quantitative metric that quantifies the extent to which the constraints (through STN_{act}) and actions of the failing plan were directly and indirectly affected by the failure after propagation. We determined $k_2 = 0.2$ as an appropriate setting, since the impact is primarily on the failing plan. Finally, S_{realloc} represents the number of robots-teams allocations that are modified after the failure is propagated through $\text{STN}_{\text{teams}}$. This provides a useful model for the reallocation impact and total replanning necessity. It also allows for further reflection of the impact variation between the different recovery mechanisms. To increase this variation, we chose $k_3 = 0.3$. This metric facilitates quantitative comparison of localized versus cascading failures and guides the selection of recovery mechanisms.

Failure Scenario

A failure affected one robot on a non-critical team (see Figure 11). The failure was propagated, but all impacted teams were reallocatable. The DEM selected a new team from idle robots, validated constraint satisfaction, and resumed execution. AMA-PLAN did not trigger a new plan.

In this scenario, a failure occurs in a single team (`Team 0`), while all other teams either remain valid or are not impacted by the failure propagation. AMA-EXEC's execution monitoring system

detects this localized failure and initiates a recovery process without triggering a full reallocation or global replanning.

This is made possible through:

- Constraint-based validation of all team subproblems.
- VAL-based plan verification to assess subplan viability and mission degradation.
- Targeted propagation of failure effects to dependent teams only.

The only directly failed team is `Team 0`, and a dependent synchronized team, `sync_0_a`, exhibits partial degradation. The rest of the mission structure remains unaffected, and all existing allocations can be preserved.

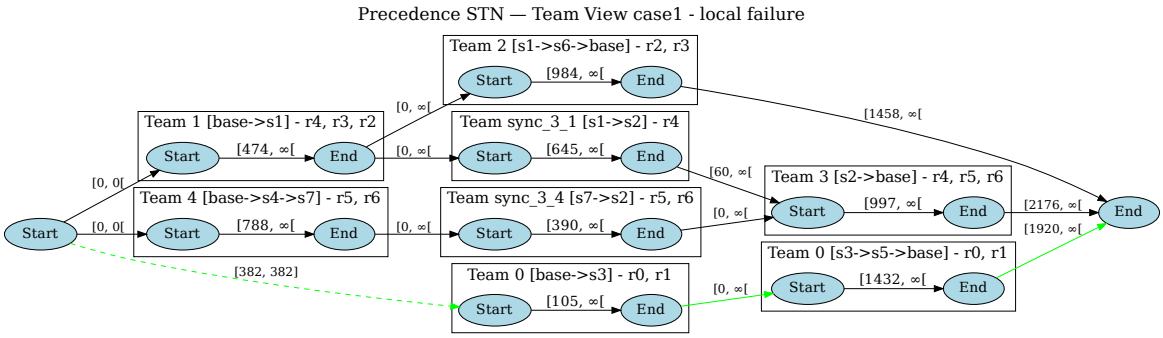


Figure 11: Reconstructed STN_{teams} for Case 1. Team 0 in Figure 7 is invalidated due to failure and is replanned (at 382 s), All other teams remain unaffected and continue execution.

6.1.3 DEM **Case 2: Localized Failure with Partial Reallocation**

A critical failure occurred during execution of `sample robot1 pp6 site3`, invalidating the current team on `Team 0` (see Figure 12). Unlike Case 1, the failure could not be resolved by minor adjustments, requiring partial team reallocation.

- `robot4` was reassigned to assist `robot0` on `Team 0`.
- `Team 3`, previously executed by `robot4` with `robot5` and `robot6`, will be done by the two robots later.
- `Team 2` remains unaffected, and global replanning was avoided.

The higher VAL and global impact scores (0.67 and 0.5) indicate a broader degradation than in Case 1, though still manageable without a full AMA-PLAN replan.

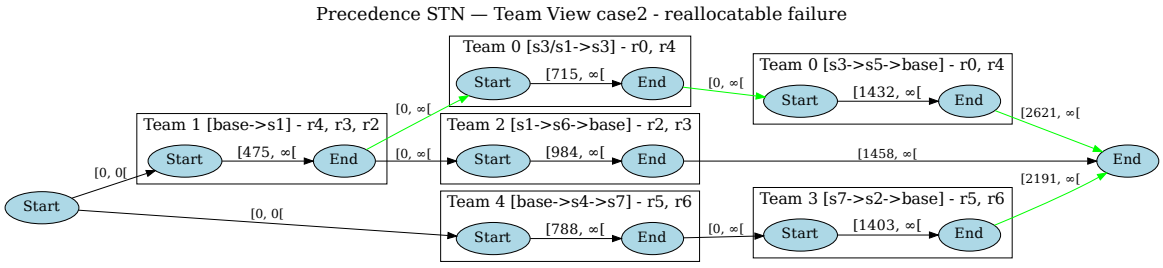


Figure 12: Reconstructed STN_{teams} for Case 2. team 0 in Figure 7 failed. Robot4 was re-allocated from team 1 to team 0 after team 1 had finished.

6.1.4 DEM **Case 3: Escalated Multi-Team Failure Triggering Full Replanning**

The next case illustrates the final mechanism: full replanning. Following the partial reallocation successfully performed in Case 2, an additional failure occurred on an unrelated team, `Team 2` (see Figure 13). This new failure, combined with existing degraded paths, invalidated the current mission configuration and required a full replanning step.

This scenario illustrates the limits of localized recovery. While the AMA-EXEC Execution Manager can absorb isolated failures through constraint-aware reallocation, compounded failures across disjoint teams trigger a complete mission reassessment and regeneration.

- The initial failure (`sample robot1 pp6 site3`) was recovered via partial reassignment on `Team 0`.
- A second failure on `Team 2` (`observe robot3 ...`) rendered its team invalid.
- Multiple teams became unrecoverable under current constraints, and AMA-PLAN was called to generate a new plan.

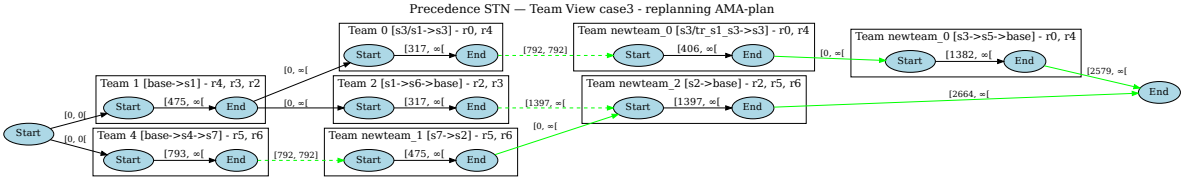


Figure 13: Reconstructed STN_{teams} for Case 3. Teams 2 failed after Case 2's Team 0 failure in Figure 12. The current teams and paths could not be continued, requiring a path replanning procedure.

Table 2 summarizes the 3 DEM recovery scenarios presented above.

Table 2: Comparative Summary of Failure Recovery Scenarios

Metric	Case 1	Case 2	Case 3
Failing Teams	0	0	2
Failed Action	`translate_data robot0 sp9 site3 B`	`sample robot1 pp6 site3`	`observe robot3 cpsite6 site6`
Failure Type	robot_cannot_relay	critical_robot_failure	2nd critical_robot_failure
Other Affected Teams	sync_0_a	sync_0_a, 3	2,3,4,0
Unaffected Teams	1, 2, 3, 4, sync_3_1, sync_3_4	1, 2, 4	1
Non-Required Teams	—	sync_3_1, sync_3_4	1
Global Replan Required	No	No	Yes
Preserved Allocation	Yes	Partial (Teams 0 and 3 reallocated)	None
VAL Global Score S_{val}	0.375	0.6667	1.0
Global Impact Score S_{global}	0.236	0.4959	0.9356
Replanning Time (s)	3.93	8.75	16.26

6.2 Comparative Evaluation and Analysis

In a second step, we evaluate the system by providing both qualitative insights into delay injection mechanisms and quantitative evaluation of AMA-PLAN planning time and distribution, and AMA-EXEC recovery mechanisms across a wide range of problems and failure scenarios. We focus on temporal supervision responsiveness, recovery mechanism selection, and the trade-offs between local reallocation and full replanning. As before, experiments were conducted on a machine with an Intel Core i7-11800H (8 cores, 16 threads, max 4.6 GHz) and 32 GB RAM, running Ubuntu.

6.2.1 Delay Injection Statistics Across Teams

We first analyzed how frequently the STN-CONTROLLER injected delays to preserve symbolic constraints under different team configurations presented in Section 6.1. Table 3 summarizes the delay statistics observed across three representative team plans executed in parallel, each varying in size and number of resulting E_C constraints bottlenecks.

Table 3: STN delay injection behavior across different teams.

Team	Duration (s)	# Delays	Avg Delay (s)	# Bottlenecks
Team 0 (2 robots)	504.80	4	16.36	3
Team 1 (3 robots)	473.85	1	7.34	2
Team 4 (2 robots)	787.97	3	5.97	4

This observation demonstrates that delay injection is not solely a function of team size and plan duration, but also of the temporal structure of the plan and action dependencies. The STN-CONTROLLER effectively adapts its intervention strategy depending on the concurrency profile, injecting minimal and targeted delays to maintain symbolic order without triggering full replanning.

6.2.2 Planning Time and Scalability Analysis

Following with the quantitative study, we first assess the runtime cost of AMA-PLAN across all tested configurations ($|\mathscr{S}| \in \{4,\ldots,20\}$, $|\mathscr{R}| \in \{4,\ldots,20\}$) with 30 batches of randomized inputs for a total of 2,430 planning problem instances. Each instance corresponds to a complete mission planning run, including symbolic decomposition and $\text{STN}_{\text{teams}}$ resolution. Higher-$|\mathscr{S}|$ scenarios contain multiple points of interest (PoIs), yielding between 25 and more than 50 sampling objectives (pollution points pp in P_{spl}). Grounded PDDL action counts routinely exceed 100, as an example taking back our 7 sites and 7 robots illustrative mission (Figure 2) achieves sampling of $|P_{spl}| = 18$ and compiles into 143 grounded actions distributed across team-specific subproblems presented in Figure 7 for a total mission time of 2,236–2,246 s depending on temporal drift during execution.

We chose 300 s as a timeout threshold as our study focuses on planning at execution level, and planning time is an important variable to avoid idle time in that context. Each PDDL subproblem was also capped at 60 s wall-clock to reflect the same constraints.

Figure 14 shows the average planning time and fraction of successful runs per configuration. Timeouts appear first for large site counts ($|\mathscr{S}| \geq 14$) and at higher $|\mathscr{R}|$ for fixed $|\mathscr{S}|$, consistent with the leading terms in our complexity bound. Most moderate-size cases (e.g., $|\mathscr{S}| \leq 10$, $|\mathscr{R}| \leq 10$) complete well below 60 s. Unsuccessful cells correspond strictly to timeout.

Figure 15 decomposes planning cost at fixed $|\mathscr{R}|$. The stacked bars show that STN solving is the dominant contributor at lower scales, while clustering and robot assignment remain comparatively minor. As $|\mathscr{S}|$ grows, per-scenario construction (sites and clusters) increases sharply and eventually overtakes STN solving, explaining the steep rise in planning times. The 95% confidence intervals (top) confirm that these averages are stable across batches. Overall, scalability degrades gracefully up to $|\mathscr{S}| \approx 12$, after which the combined growth of scenario construction and STN solving causes some runs to approach or exceed the 300 s timeout. We therefore treat the 300 s planning cutoff as our practical real-time deployment threshold: configurations exceeding this limit are considered operationally infeasible due to the idle time they would induce during execution.

6.2.3 Runtime Failure Recovery Analysis

We now turn to the evaluation of AMA-EXEC under failure. We categorize the failures by failure types:

- **Failure type 1:** a change in the environment (for example, unusable relay point).
- **Failure type 2:** robot loses capability (e.g., relay, sample);
- **Failure type 3:** robot becomes unavailable (crash);

and failures are injected at different points in the $\text{STN}_{\text{teams}}$:

- **Strategy 1 (Early):** in a team with no predecessor;

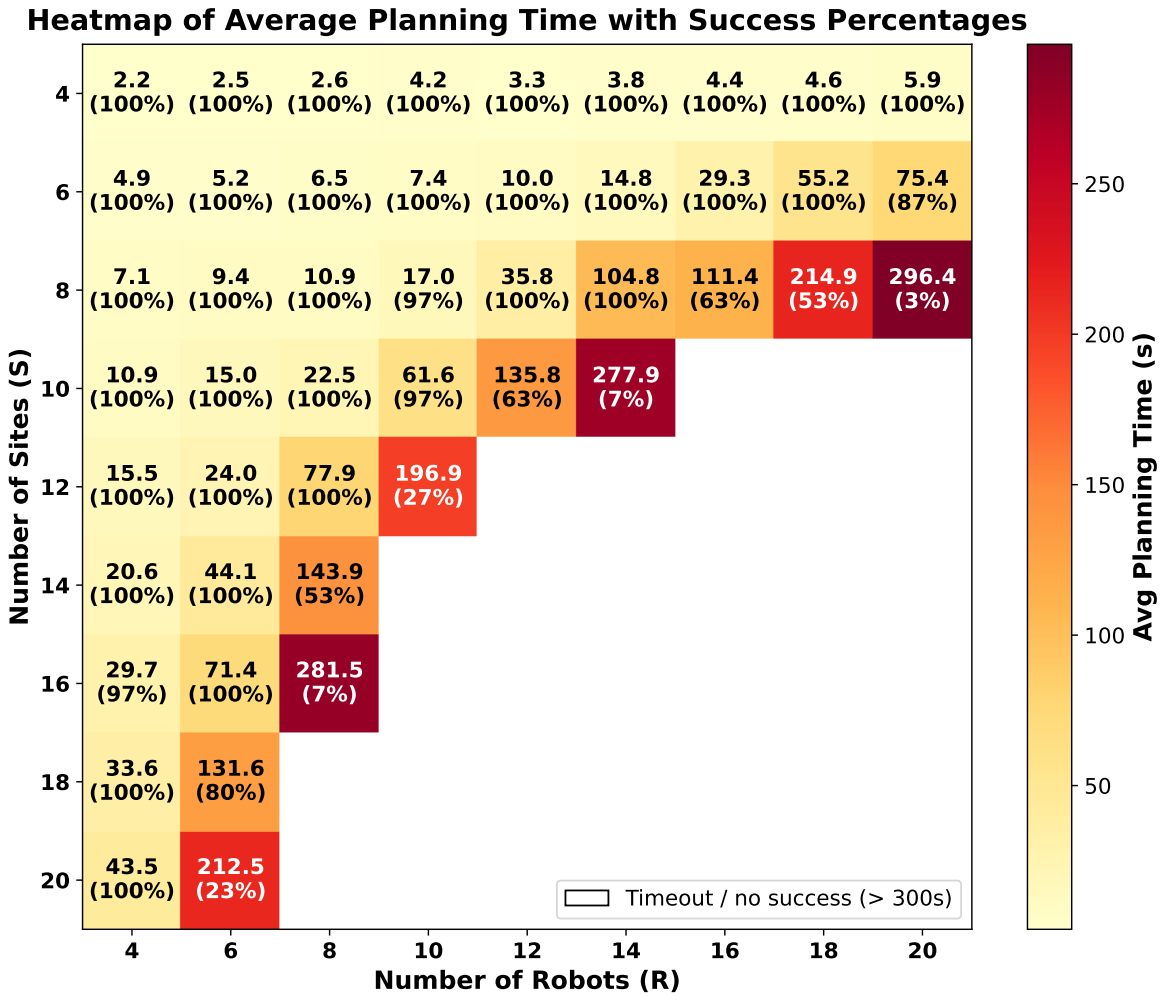


Figure 14: Heatmap of average planning time (seconds) with success rate (parentheses). Each cell aggregates 30 batches. White borders denote at least one run exceeding the 300 s timeout.

- **Strategy 2 (Mid):** in a team with both predecessors and successors;
- **Strategy 3 (Late):** in a terminal team with at least one predecessor.

A total of 9,720 failure cases were tested, covering all $(|\mathscr{S}|, |\mathscr{R}|)$ configurations with 30 randomized batches each, as well as all failure types and strategies. For each failure, we recorded the recovery mechanism selected by AMA-EXEC and its estimated cost. As described in Section 6.1, AMA-EXEC can recover by local symbolic replanning, by reallocating robots within $\text{STN}_{\text{teams}}$, or by full replanning. Across all runs, missions completed unless planning timed out; we did not observe aborts, since the current quantitative failure tests do not account for high-scale cascading failures or enforce a strict makespan bound that could render a mission unsolvable.

Figure 16 confirms that the nature of the failure significantly influences recovery behavior. Robot crashes will have a higher impact on the plans and often require re-allocation or full re-

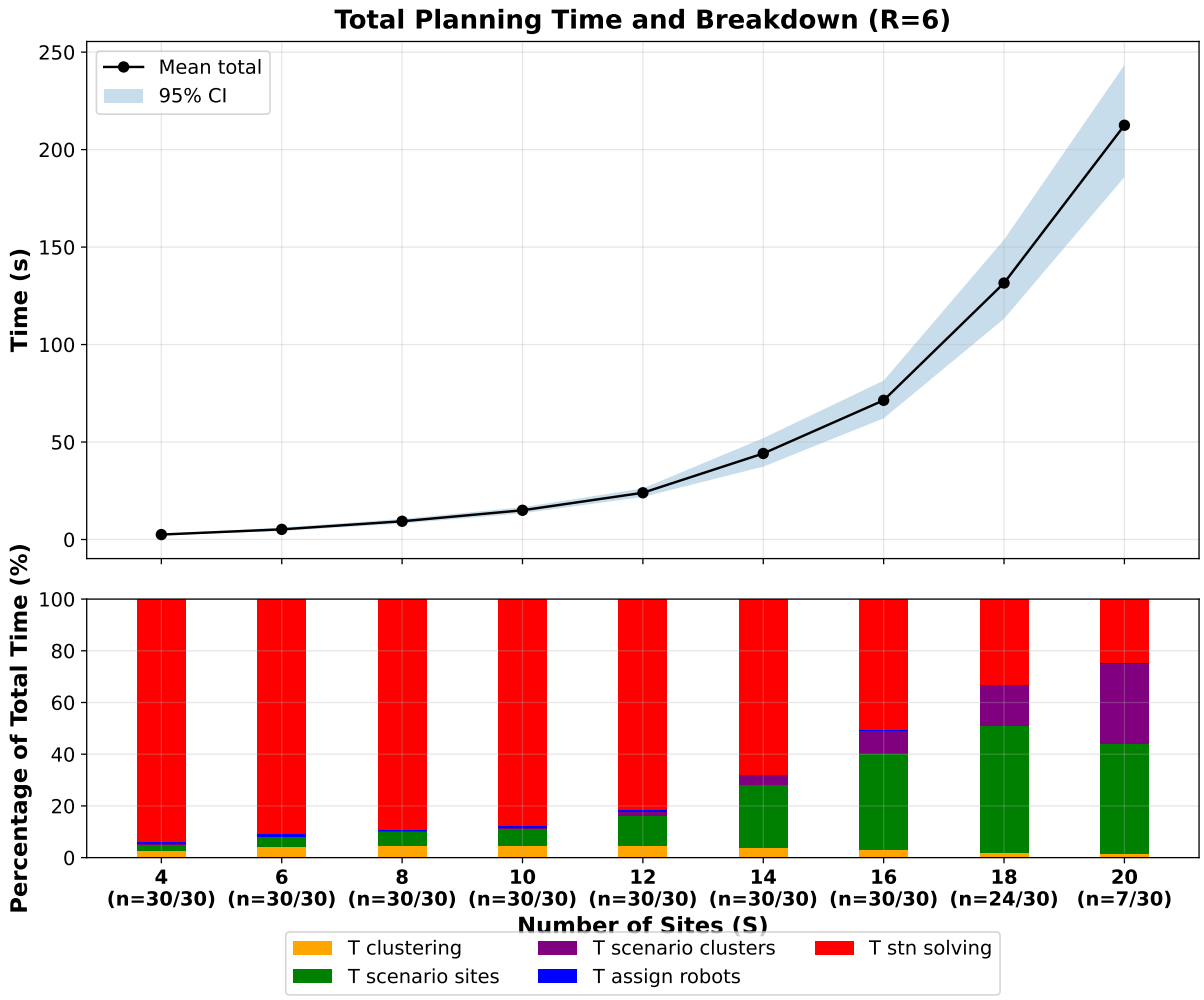


Figure 15: Total planning time and component breakdown for $|\mathscr{R}| = 6$. Top: mean total planning time with 95% bootstrap confidence intervals. Bottom: percentage of time per component (clustering, site-level construction, cluster-level construction, robot assignment, STN creation and solving).

planning (only instances where more than two robots were allocated on the failing team might recover through simple local replanning). On the other hand, robot losing capability and environment changes usually do not lead to severe failure impact, as a simple switch of assigned role or location of action could allow for local replanning.

Finally, as shown in Figure 17, most recovery mechanisms are completed significantly faster, and most failures lead to local replanning of the failed team. Our failure handling strategy demonstrates high adaptability and robust recovery output for various failure types while recovery times remain below full mission re-synthesis, validating the modular and adaptive architecture.

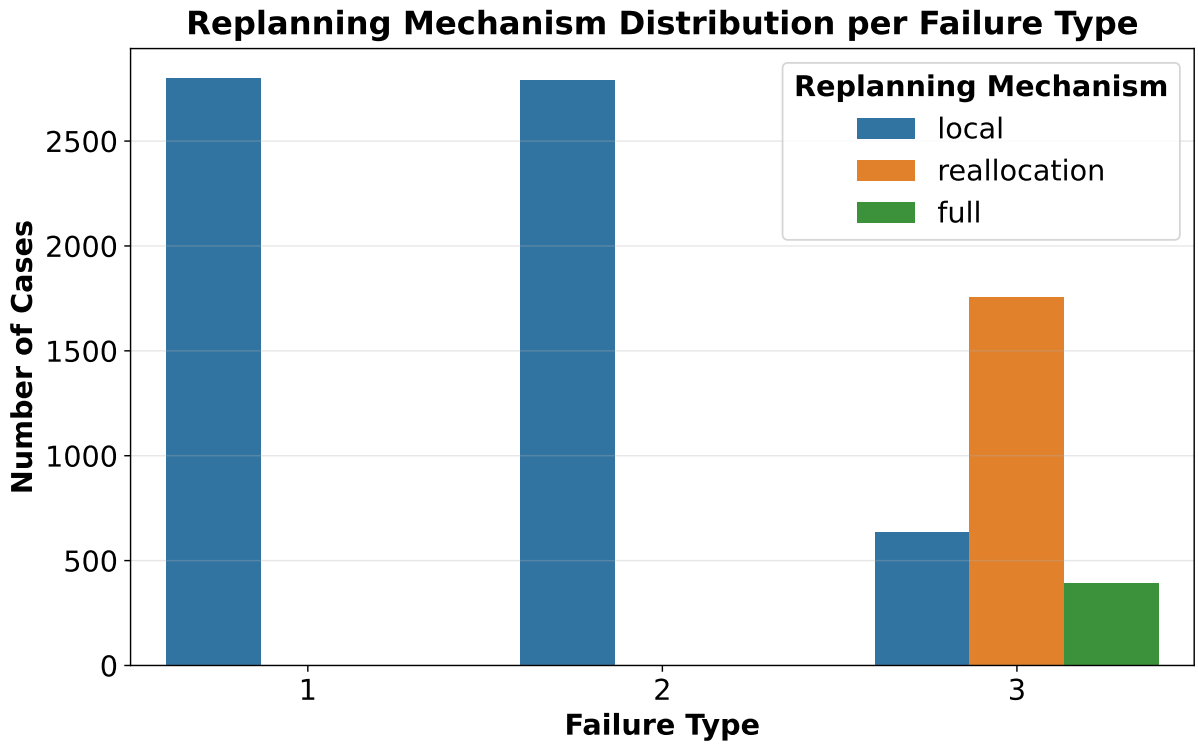


Figure 16: Replanning mechanism distribution per failure type. Robot crashes (Failure Type 3) mostly trigger reallocation. Robot's capability loss (Failure Type 2) or environmental changes (Failure Type 1) usually leads to local replanning.

6.3 Toward Real-World Deployment

While our evaluation is simulation-based, the simulator mirrors deployed ROS 2 interfaces (actions, topics, services; PlanSys2-compatible), and execution parameters were derived from platform specifications and field observations. AMA-EXEC emphasizes executability: the STN-CONTROLLER absorbs runtime drift via conservative margins and delay injection, and the DEM localizes replanning to the minimum affected scope. These mechanisms target field realities such as network delays and packet loss (which can desynchronize collaborative actions), sensor noise, partial capability loss (e.g., sampling or relay), and asymmetric costs at air–water transitions. AMA acts as a supervisory layer on top of perception and navigation, maintaining symbolic and temporal coherence even when lower-level behaviors are imperfect.

7 Conclusion

In this work, we introduced AMA, a modular and distributed architecture designed to enable robust mission execution in complex multi-robot environments, such as multi-medium scenarios. Combined with AMA-PLAN, AMA-EXEC guarantees the execution of temporally annotated

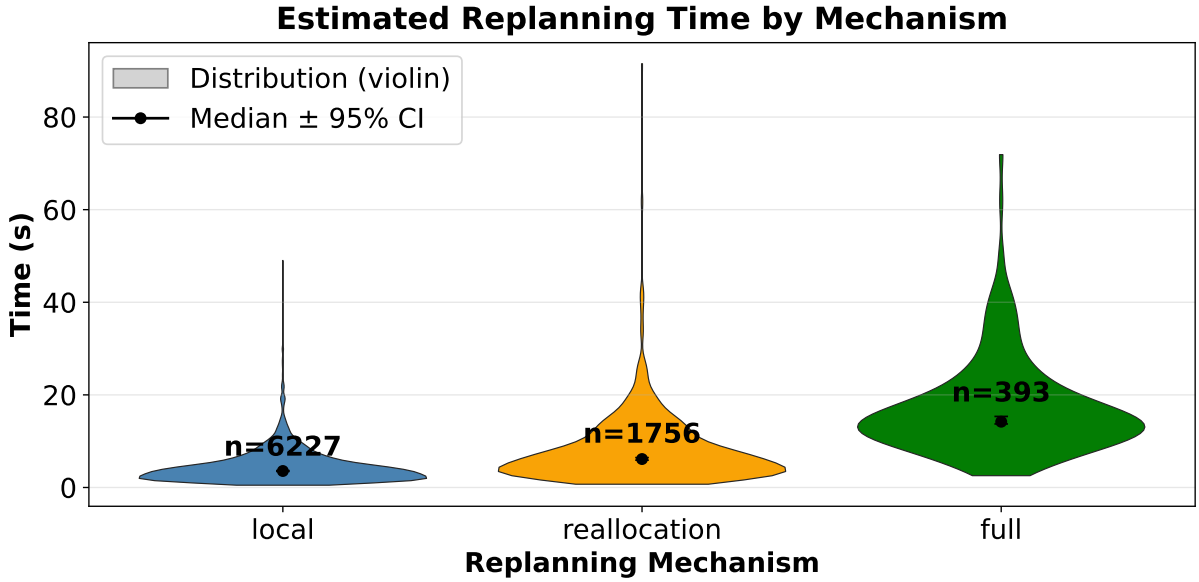


Figure 17: Distribution of replanning times across 9,720 injected failures, shown per mechanism as violin plots with median and confidence intervals. Local replanning typically completes within $\leq$5–10 s, reallocations take slightly longer but remain mostly under 20 s, while full replanning exhibits the longest tail and more occasional high-latency cases.

PDDL plans with resilience to drifts, delays, and failures. The integration of STN-based supervision, the extension of BT execution in distributed environments, and fast, constraint-aware recovery enable AMA-EXEC to manage complex scenarios without necessitating full mission replanning while preserving execution of unaffected teams. Our experimental results demonstrate that AMA-EXEC significantly extends the capabilities of the PlanSys2 execution system by maintaining symbolic and temporal coherence across teams, even in the presence of execution failures.

Future work will pursue several directions. First, we aim to validate AMA-EXEC on more advanced simulation platforms integrating a hybrid aquatic–aerial world simulator. Second, we plan to extend its modularity by integrating the recent STN–BT builder addition to PlanSys2, reinforcing temporal robustness and mission control. Finally, we intend to apply AMA-EXEC and AMA-PLAN to domains beyond trans-media missions, such as heterogeneous robotic teams, platform inspection, and large-scale exploration.

References

J. F. Allen. An Interval-Based Representation of Temporal Knowledge. In *International Joint Conference on Artificial Intelligence*, 1981. URL http://www.ijcai.org/Past%20Proceedings/IJCAI-81-VOL%201/PDF/045.pdf.

A. Belbachir, F. Ingrand, and S. Lacroix. A cooperative architecture for target localization using multiple AUVs. *Intelligent Service Robotics*, 5(2):119–132, Apr. 2012. doi:10.1007/s11370-012-0107-1.

J. Benton, A. Coles, and A. Coles. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 2–10, May 2012. URL https://dl.acm.org/doi/10.5555/3038546.3038548.

L. Bramblett, B. Miloradovic, P. Sherman, A. V. Papadopoulos, and N. Bezzo. Robust online epistemic replanning of multi-robot missions. *arXiv preprint*, 2024. doi:10.48550/arXiv.2403.00641.

M. Brenner. A multiagent planning language. *ICAPS-2003 workshop on PDDL*, 06 2003. URL https://icaps03.icaps-conference.org/satellite_events/documents/WS/ws3/05/brenner.pdf.

M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, June 2009. doi:10.1007/s10458-009-9081-1.

Y. Carreno, E. Pairet, Y. Petillot, and Ronald P. A. Petrick. Task Allocation Strategy for Heterogeneous Robot Teams in Offshore Missions. In *Autonomous Agents and MultiAgent Systems*, Auckland, New Zealand, 05 2020. doi:10.5555/3398761.3398792.

M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, A. Carrera, N. Palomeras, N. Hurtos, and M. Carreras. ROSPlan: Planning in the Robot Operating System. In *International Conference on Automated Planning and Scheduling*, 2015. doi:10.1609/icaps.v25i1.13699.

M. Colledanchise and P. Ögren. *Behavior Trees in Robotics and AI*. CRC Press, 2018. doi:10.1201/9780429489105.

M. Colledanchise, D. Almeida, and P. Ögren. Towards blended reactive planning and acting using behavior trees. In *International Conference on Robotics and Automation (ICRA)*, pages 8839–8845. IEEE, 2019. doi:10.1109/ICRA.2019.8794128.

V. De La Rochefoucauld, S. Lacroix, P. Ratsamee, and H. Takemura. Solving multi-robot task allocation and planning in trans-media scenarios. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5764–5769. IEEE, 2024. doi:10.1109/IROS58592.2024.10801394.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3): 61–95, 1991. doi:10.1016/0004-3702(91)90006-6.

M. Di Rocco, F. Pecora, and A. Saffiotti. When robots are late: Configuration planning for multiple robots with dynamic goals. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013. doi:10.1109/IROS.2013.6697214.

M. D'Inverno, M. Luck, M. Georgeff, D. Kinny, and M. Wooldridge. The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):5–53, 2004. doi:10.1023/B:AGNT.0000019688.11109.19.

A. Finzi, F. Ingrand, and N. Muscettola. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004. doi:10.1109/IROS.2004.1389463.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, Dec. 2003. ISSN 1076-9757. doi:10.1613/jair.1129.

B. P. Gerkey and M. J. Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, Sept. 2004. doi:10.1177/0278364904045564.

M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld. PDDL - The Planning Domain Definition Language. Technical report, AIPS, 08 1998. URL `https://engineering.purdue.edu/~relation/surf/papers/pddl.pdf`.

M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. doi:10.1017/CBO9781139583923. URL `https://projects.laas.fr/planning/`.

R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule. Behavior trees and state machines in robotics applications. *IEEE Transactions on Software Engineering*, 49(9):4243–4267, 2023. doi:10.1109/TSE.2023.3269081.

R. Howey, D. Long, and M. Fox. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. In *International Conference on Tools with Artificial Intelligence*, pages 294–301, 2004. doi:10.1109/ICTAI.2004.120.

F. Ingrand, S. Lacroix, S. Lemai-Chenevier, and F. Py. Decisional autonomy of planetary rovers. *Journal of Field Robotics*, 24(7):559–580, 2007. doi:10.1002/rob.20206.

M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith. A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems*, 154, 2022. doi:10.1016/j.robot.2022.104096.

S. Y. Jeon, I. S. Yang, B. H. Chi, H. U. Yoo, and H. R. Choi. Behavior tree-based task planning for multiple mobile robots using a data distribution service. *Electronics*, 11(4):601, 2022. doi:10.3390/electronics11040601.

S. Joyeux, R. Alami, S. Lacroix, and R. Philippsen. A plan manager for multi-robot systems. *The International Journal of Robotics Research*, 28(2):220, 2009. doi:10.1177/0278364908098370.

D. L. Kovacs. A multi-agent extension of PDDL3.1. In *The International Planning Competition*, pages 19–27, 2012. URL https://repozitorium.omikk.bme.hu/server/api/core/bitstreams/22c3b17f-5979-4170-9f0d-a6798800ca23/content.

W. Li, S. Yan, L. Shi, J. Yue, M. Shi, B. Lin, and K. Qin. Multiagent consensus tracking control over asynchronous cooperation–competition networks. *IEEE Transactions on Cybernetics*, 55 (9):4347–4360, 2025a. doi:10.1109/TCYB.2025.3583387.

W. Li, J. Yue, M. Shi, B. Lin, and K. Qin. Neural network-based dynamic target enclosing control for uncertain nonlinear multi-agent systems over signed networks. *Neural Networks*, 184:107057, 2025b. ISSN 0893-6080. doi:10.1016/j.neunet.2024.107057.

F. Martín, J. G. Clavero, V. Matellán, and F. J. Rodríguez. PlanSys2: A planning system framework for ROS2. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021. doi:10.1109/IROS51168.2021.9636544.

F. Martín, M. Morelli, H. Espinoza, F. J. Rodríguez-Lera, and V. Matellán Olivera. Optimized execution of PDDL plans using behavior trees. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 1596–1598, 2021. doi:10.5555/3463952.3464171.

A. Micheli, A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini, L. Iocchi, F. Ingrand, U. Köckemann, F. Patrizi, A. Saetti, I. Serina, and S. Stock. Unified planning: Modeling, manipulating and solving AI planning problems in python. *SoftwareX*, 29:102012, 2025. doi:10.1016/j.softx.2024.102012.

R. Olfati-Saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007. doi:10.1109/JPROC.2006.887293.

S. Patra, M. Ghallab, D. S. Nau, and P. Traverso. APE: An Acting and Planning Engine. *Advances in Cognitive Systems*, pages 1–20, Sept. 2019. URL https://univ-tlse2.hal.science/hal-01959098v1.

S. Patra, J. Mason, M. Ghallab, D. Nau, and P. Traverso. Deliberative acting, planning and learning with hierarchical operational models. *Artificial Intelligence*, 299, 2021. doi:10.1016/j.artint.2021.103523.

A. A. Pedroso, A. C. Da Silva, P. M. Pinheiro, and P. L. J. Drews. Prototyping and Construction of a Hybrid Unmanned Aerial Underwater Vehicles. In *Latin American Robotics Symposium (LARS)*, pages 61–66, São Bernardo do Campo, Brazil, Oct. 2022. IEEE. doi:10.1109/LARS/SBR/WRE56824.2022.9995873.

A. Polyakov. Nonlinear feedback design for fixed-time stabilization of linear control systems. *IEEE Transactions on Automatic Control*, 57(8):2106–2110, 2012. doi:10.1109/TAC.2011.2179869.

P. Ratsamee, P. Tempattarachoke, L. Jirachuphun, M. Miwa, and K. Somprasong. Point cloud estimation during aerial-aquatic transition in monocular camera-based localization and mapping. *Journal of Robotics and Mechatronics*, 35(6):1645–1654, 2023. doi:10.20965/jrm.2023.p1645.

A. Reina, G. Valentini, C. Fernández-Oto, M. Dorigo, and V. Trianni. A design pattern for decentralised decision making. *PLOS ONE*, 10(10):1–18, October 2015. doi:10.1371/journal.pone.0140950.

W. Ren and R. Beard. *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1849967016. doi:10.1007/978-1-84800-015-5.

A. Schweim, M. Zager, M. Schweim, A. Fay, and J. Horn. Unmanned vehicles on the rise: a review on projects of cooperating robot teams. *at - Automatisierungstechnik*, 72(1):3–14, 2024. doi:10.1515/auto-2022-0153.

F. Shkurti, A. Xu, M. Meghjani, J. C. Gamboa Higuera, Y. Girdhar, P. Giguère, B. B. Dey, J. Li, A. Kalmbach, C. Prahacs, K. Turgeon, I. Rekleitis, and G. Dudek. Multi-domain monitoring of marine environments using a heterogeneous robot team. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012. doi:10.1109/IROS.2012.6385685.

J. Stedl and B. C. Williams. Managing communication limitations in partially controllable multi-agent plans. In *ICAPS Workshop on Multiagent Planning and Scheduling*, pages 8–14, 2005. URL `https://groups.csail.mit.edu/mers/old-site/papers/HRA_ICAPS_Workshop.pdf`.

Y. H. Tan and B. M. Chen. Design of a Morphable Multirotor Aerial-Aquatic Vehicle. In *OCEANS*, pages 1–8, Seattle, WA, USA, Oct. 2019. IEEE. doi:10.23919/OCEANS40490.2019.8962867.

Y. H. Tan and B. M. Chen. Survey on the Development of Aerial–Aquatic Hybrid Vehicles. *Unmanned Systems*, 09(03):263–282, July 2021. doi:10.1142/S2301385021410028.

J. Turi and A. Bit-Monnot. Extending a refinement acting engine for fleet management: Concurrency and resources. In *International Conference on Tools with Artificial Intelligence (ICTAI)*, 2022. doi:10.1109/ICTAI56018.2022.00216.

A. Umbrico, A. Cesta, M. Cialdea Mayer, and A. Orlandini. PLATINUm: A New Framework for Planning and Acting. In *AI*IA*. Springer International Publishing, 2017. doi:10.1007/978-3-319-70169-1_37.

A. Vasilijevic, D. Nad, F. Mandic, N. Miskovic, and Z. Vukic. Coordinated Navigation of Surface and Underwater Marine Robotic Vehicles for Ocean Sampling and Environmental Monitoring. *IEEE/ASME Transactions on Mechatronics*, 22(3), June 2017. doi:10.1109/TMECH.2017.2684423.

H. Wang, Q. Liu, and J. H. Park. Predefined-time fuzzy adaptive optimal secure consensus control for multiagent systems with unknown follower dynamics. *IEEE Transactions on Fuzzy Systems*, 33(7):2122–2135, 2025. doi:10.1109/TFUZZ.2025.3552050.

W. Wang, C. Wen, J. Huang, and J. Zhou. Adaptive consensus of uncertain nonlinear systems with event triggered communication and intermittent actuator faults. *Automatica*, 111:108667, 2020. ISSN 0005-1098. doi:10.1016/j.automatica.2019.108667.

G. Yao, Y. Li, H. Zhang, Y. Jiang, T. Wang, F. Sun, and X. Yang. Review of hybrid aquatic-aerial vehicle (HAAV): Classifications, current status, applications, challenges and technology perspectives. *Progress in Aerospace Sciences*, 139, May 2023. doi:10.1016/j.paerosci.2023.100902.

J. Zapf, M. Roveri, F. Martin, and J. C. Manzanares. Constructing behavior trees from temporal plans for robotic applications. *arXiv preprint*, 2024. doi:10.48550/arXiv.2406.17379.