

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Design of a Software Tool and Methodology for Benchmarking a Fleet Coordination System

Diogo Soares de Albergaria Oliveira



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Pedro Luís Cerqueira Gomes da Costa

July 30, 2026

Resumo

A adoção em larga escala de Robots Móveis Autónomos (AMRs) na intralogística tornou a coordenação de frotas um desafio operacional central, onde o congestionamento e os impasses em estrangulamentos estruturais devem ser prevenidos ao longo de toda a cadeia de alocação de tarefas, planeamento de trajetórias e execução supervisionada. Determinar se um sistema de coordenação satisfaz estas exigências requer uma avaliação sistemática e reproduzível; contudo, as plataformas de benchmarking existentes avaliam algoritmos de Multi-Agent Path Finding em mapas abstratos e sintéticos ou medem métricas isoladas, não capturando indicadores de desempenho transversais às camadas de planeamento, execução e replaneamento numa frota AMR real. Esta dissertação desenvolve uma metodologia e o software correspondente para colmatar essa lacuna. Um pipeline ROS automatizado lança, monitoriza e termina cada execução em condições idênticas; um nó `mission_benchmark` regista cada execução num esquema YAML estruturado, incluindo o histórico de planos por missão; uma baseline de semáforo fornece uma referência reativa sem antecipação temporal; e uma ferramenta de análise agrega os registos em vistas comparativas de escalabilidade, fidelidade de plano, replaneamento e segurança. A metodologia é demonstrada comparando o coordenador de referência com consciência temporal (TEA*) com a baseline de semáforo num layout industrial aberto e num layout hospitalar condicionado, sob variação sistemática do tamanho da frota e da carga de trabalho, onde o TEA* mantém um envelope operacional substancialmente mais alargado enquanto o semáforo colapsa em impasse à medida que a densidade aumenta. Uma vez que o pipeline, o esquema de dados e a ferramenta de análise dependem apenas de um conjunto padronizado de tópicos ROS e não de nenhum coordenador em particular, são reutilizáveis pela equipa do INESC TEC para quantificar o impacto de iterações futuras do coordenador.

Abstract

The large-scale adoption of Autonomous Mobile Robots (AMRs) in intralogistics has made fleet coordination a central operational challenge, where congestion and deadlocks at structural bottlenecks must be prevented across the full pipeline of task allocation, path planning, and supervised execution. Establishing whether a coordination system meets these demands requires systematic and reproducible evaluation, yet existing benchmarking platforms assess Multi-Agent Path Finding algorithms on abstract, synthetic maps or measure isolated metrics, and none captures cross-layer key performance indicators spanning planning, execution, and replanning on a real AMR fleet. This dissertation develops a methodology and the accompanying software to close that gap. An automated ROS pipeline launches, monitors, and terminates every run under identical conditions; a `mission_benchmark` node records each run as a structured YAML schema, including per-mission plan history; a semaphore baseline provides a reactive reference with no temporal look-ahead; and an analysis tool aggregates the records into comparative views of scalability, plan fidelity, replanning, and safety. The methodology is demonstrated by comparing the reference time-aware coordinator (TEA*) against the semaphore baseline across an open industrial layout and a constrained hospital layout, under systematic variation of fleet size and workload, where TEA* sustains a substantially wider operational envelope while the semaphore collapses into deadlock as density rises. Because the pipeline, data schema, and analysis tool depend only on a standardised set of ROS topics and not on any particular coordinator, they are reusable by the INESC TEC team to quantify the impact of future iterations of the coordinator.

UN Sustainable Development Goals

This dissertation contributes to the advancement of AMRs fleet coordination in intralogistics environments. By designing a rigorous benchmarking methodology and software tool for evaluating fleet coordination systems, this work supports the development of more efficient, reliable, and scalable industrial automation.

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

SDG	Target	Contribution	Performance Indicators and Metrics
8	8.2	Developing a benchmarking tool for AMR fleet coordination enables the systematic identification of more efficient coordination strategies supporting higher productivity.	Reduction in fleet mission completion time and solution cost across evaluated coordination strategies.
9	9.4	The benchmarking methodology promotes the adoption of more efficient AMR coordination systems by quantifying the impact of design choices on deadlocks, congestion, and replanning overhead in intralogistics infrastructure.	Reduction in failed planning attempts and supervision-triggered replanning events; improvement in real execution time.
	9.b	Supporting the evaluation of the IN-ESC TEC time-aware fleet coordination stack contributes to domestic technological development and strengthens research capacity in industrial automation and multi-robot systems.	Demonstrated performance advantage of the time-aware reference coordinator over simpler baseline strategies across benchmarked bottleneck scenarios.

Acknowledgements

I would like to begin by thanking my supervisor, Prof. Pedro Costa, for his guidance and availability throughout this dissertation. His support in the direction of the work and his feedback at every stage were invaluable.

I am equally grateful to my co-supervisor, Diogo Matos, whose technical expertise and daily involvement were essential to the development of this project. His patience in answering every question and his dedication to this work made a real difference.

To Afonso, my university friend and companion throughout this degree, thank you for the advice, the patience, the shared frustrations, and for always knowing how to help me. Having someone to navigate this journey with made everything easier.

To my parents, António and Alexandra, and my sister Carolina, thank you for everything. For the support, for always believing in me, for trusting me, and for never letting me settle for less than my best. Everything I have achieved is because of you, and this dissertation would not have been possible without you.

To my girlfriend Mariana, thank you for your patience, your support, and for being by my side every day, especially the less happy ones. The simplicity and ease you bring to my life is what made everything feel possible, and none of this would have been the same without you.

To handball, and to the people it brought into my life, thank you for the balance. The training sessions, the games, and the friendships built along the way gave me the space to breathe and the energy to keep going. Most of my closest friends came through handball, and this degree was completed with them alongside me.

Diogo Oliveira

Contents

1	Introduction	1
1.1	Motivation and Context	1
1.2	Problem Statement	2
1.3	Benchmarking Gap	3
1.4	Objectives and Scope	4
1.5	Dissertation Structure	5
2	Background and Fundamental Aspects	6
2.1	Fleet Coordination Architecture	6
2.2	MAPF Definitions: Conflicts, Objectives, and Temporal Variants	7
2.3	Coordination Paradigms	8
2.4	Reference System	9
2.5	Summary	11
3	State of the Art	12
3.1	Real-world fleet coordination in intralogistics and warehouses	12
3.2	Task allocation and scheduling in robot fleets	13
3.2.1	Families of approaches	13
3.2.2	Coupling with path planning and coordination	14
3.2.3	Evaluation practice and missing pieces	15
3.3	MAPF and planning methods for coordination	15
3.3.1	Solver families and coordination paradigms	15
3.3.2	Key trade-offs for intralogistics coordination	17
3.3.3	Evaluation practice and gaps (planning vs. execution)	17
3.4	Execution uncertainty, supervision, and online replanning	17
3.4.1	Disturbance models	18
3.4.2	Supervision and monitoring mechanisms	18
3.4.3	Online replanning and recovery strategies	20
3.4.4	Evaluation practice and remaining gaps	20
3.5	Benchmarking frameworks and evaluation methodologies	21
3.5.1	Benchmark families	21
3.5.2	Criteria for comparative analysis of benchmarking tools	22
3.5.3	Implications for the benchmark design	22
3.6	Synthesis: Limitations and research gap	24
4	Benchmarking Methodology	26
4.1	Experimental Protocol	26
4.1.1	Independent Variables	26

4.1.2	Dependent Variables	27
4.1.3	Controlled Variables	27
4.2	Test Scenarios	28
4.3	Scientific Hypotheses	28
5	System Implementation	30
5.1	Automated ROS Benchmarking Pipeline	30
5.1.1	Pipeline Architecture	30
5.1.2	Run Lifecycle	31
5.1.3	Output Structure	32
5.2	The Node Analyzer	32
5.2.1	Dynamic Robot Discovery	32
5.2.2	KPI Computation	33
5.3	Semaphore Baseline	33
5.3.1	Operating Principle	33
5.3.2	Absence of Temporal Look-ahead and Deadlock Susceptibility	34
5.3.3	Integration with the Benchmarking Pipeline	34
5.4	Benchmarking Analysis Tool	34
5.4.1	Architecture	34
5.4.2	Interface Structure	35
5.4.3	Analytical Views	35
6	Experimental Results	41
6.1	Experimental Setup	41
6.1.1	Experimental Conditions	41
6.2	Scalability Results	42
6.2.1	Industrial Facility Scenario	42
6.2.2	Hospital Facility Scenario	43
6.3	Plan Fidelity Results	44
6.3.1	Industrial Facility Scenario	45
6.3.2	Hospital Facility Scenario	45
6.4	Replanning Analysis	46
6.4.1	Industrial Facility Scenario	46
6.4.2	Hospital Facility Scenario	47
6.5	Safety Results	48
6.5.1	Industrial Facility Scenario	48
6.5.2	Hospital Facility Scenario	50
7	Discussion	53
7.1	The Benchmark Tool	53
7.2	Scalability: Time-Aware Planning Bounds Execution Growth	54
7.3	Bottleneck Amplification: Structural Contention Widens the Gap	55
7.4	Replanning Overhead: Bounded Recovery, Not Fragility	56
7.5	Plan Fidelity: The Logged Planned Cost Underestimates Real Execution	57
7.6	Safety: Recoverable Planning Failures versus Irrecoverable Deadlock	58
7.7	Methodological Contributions and Limitations	58
7.7.1	Methodological Contributions	58
7.7.2	Limitations of the Study	59

8	Conclusions and Future Work	61
8.1	Summary of Contributions	61
8.2	Future Work	62
	References	64
A	Supplementary Data	66
A.1	Real-Time Monitoring Interface	66
A.2	Analysis Tool	67
A.3	Example Benchmark Output (YAML)	67
A.4	Plan Fidelity Ratio Tables	69
A.5	Replanning Detail Tables	70

List of Figures

2.1	Illustration of common MAPF conflict types (edge, vertex, following, cycle, swapping). Source: [9].	7
2.2	Conceptual illustration of fleet coordination architectures.	8
2.3	Example of an environment graph used for fleet coordination. Source: [10]. . . .	10
2.4	Example of encoding an infrastructure interaction/action in the graph notation. Adapted from [10].	10
4.1	Topological graphs of the two test scenarios overlaid on their respective floor plans. Blue circles represent graph nodes; edges encode navigable paths with temporal costs.	28
5.1	Architecture of the automated benchmarking pipeline.	31
5.2	Home screen of the benchmarking analysis tool, presenting the two analysis modes.	35
5.3	Quick Compare: global KPI summary table and per-robot bar charts.	36
5.4	Quick Compare: per-mission execution breakdown with execution time and cost comparison.	36
5.5	Quick Compare: per-robot comparison table with efficiency and replanning differences.	37
5.6	Detailed Analysis: Overview tab presenting global aggregate KPIs.	37
5.7	Detailed Analysis: Scalability tab visualising execution time and replanning trends against fleet size.	38
5.8	Detailed Analysis: Plan Fidelity tab showing the correlation between planned cost and execution time.	39
5.9	Detailed Analysis: Replanning tab showing the distribution of replanning load across the fleet.	39
5.10	Detailed Analysis: Safety tab comparing the operational success rate of both systems.	40
6.1	Total execution time (s) vs fleet size N in the industrial facility scenario. TEA* (System B), semaphore (System A). Missing points indicate FAILED runs.	43
6.2	Total execution time (s) vs fleet size N in the hospital facility scenario. TEA* (System B), semaphore (System A). Missing points indicate FAILED runs.	44
6.3	Planned cost vs real execution time for all missions in the industrial facility scenario. Semaphore (System A), TEA* (System B). Dashed line: ideal ratio $y = x$	45
6.4	Planned cost vs real execution time for all missions in the hospital facility scenario. Semaphore (System A), TEA* (System B). Dashed line: ideal ratio $y = x$	46
6.5	Average total replanning events vs fleet size N . TEA* (System B) and semaphore (System A).	46

6.6	Average replanning events per robot across all runs and missions. TEA* (System B) and semaphore (System A).	47
6.7	Average total replanning events vs fleet size N . TEA* (System B) and semaphore (System A).	47
6.8	Average replanning events per robot across all runs and missions. TEA* (System B) and semaphore (System A).	48
6.9	Average failed planning events vs fleet size N . TEA* (System B), semaphore (System A).	49
6.10	Operational success rate (%) vs fleet size N . Each point represents the fraction of completed runs at that fleet size. TEA* (System B), semaphore (System A). . . .	50
6.11	Average failed planning events vs fleet size N . TEA* (System B), semaphore (System A).	51
6.12	Operational success rate (%) vs fleet size N . Each point represents the fraction of completed runs at that fleet size. TEA* (System B), semaphore (System A). . . .	52
A.1	Real-time KPI dashboard active during a run.	66
A.2	Loader screen of the Detailed Analysis mode, with two folder pickers for loading datasets.	67
A.3	Fidelity ratio per robot, industrial facility scenario. System A: semaphore coordinator, System B: TEA*.	70
A.4	Fidelity ratio per robot, hospital facility scenario. System A: semaphore coordinator, System B: TEA*.	70
A.5	Average replanning events per robot, industrial facility scenario. System A: semaphore coordinator, System B: TEA*.	71
A.6	Average replanning events per robot, hospital facility scenario. System A: semaphore coordinator, System B: TEA*.	71

List of Tables

3.1	Comparative analysis of principal supervision mechanisms for multi-robot coordination.	19
3.2	Coverage of representative benchmarking platforms across coordination layers and realism assumptions.	22
6.1	Total execution time (seconds) in <code>sim1_portic</code>	42
6.2	Total execution time (seconds) in <code>sim2_hosp</code>	44

List of Acronyms

AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
CBS	Conflict-Based Search
FIFO	First-In, First-Out
ICBS	Improved Conflict-Based Search
INESC TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
KPI	Key Performance Indicator
MAPF	Multi-Agent Path Finding
MCP	Minimal Communication Policy
MRTA	Multi-Robot Task Allocation
NP	Non-deterministic Polynomial-time
OAP	Optimal Assignment Problem
PBS	Priority-Based Search
PP	Prioritized Planning
RHCR	Rolling-Horizon Collision Resolution
RMF	Robotics Middleware Framework
ROS	Robot Operating System
RPP	Revised Prioritized Planning
TEA*	Time-Enhanced A*
TRL	Technology Readiness Level
YAML	YAML Ain't Markup Language

Chapter 1

Introduction

1.1 Motivation and Context

Intralogistics environments (factories, warehouses, and hospitals) have evolved from the early use of Automated Guided Vehicles (AGVs), used in warehouses since the 1950s and primarily for transporting very large and heavy objects, towards smaller, cheaper, and more capable mobile robots enabled by advances in wireless communication, computing power, and robotic components. [1]

This transition has supported the large-scale adoption of flexible AMRs in modern warehouses, where goods-to-person systems use robots to bring inventory shelves to stationary human workers. In such deployments, productivity can increase by a factor of two or more, and large installations can require fleets of hundreds of robots operating in the same facility [1]. As warehouse operations become continuous and goals are assigned repeatedly over time, the coordination problem extends to lifelong settings, where robots are constantly given new destinations while avoiding collisions [2].

At this scale, coordinating robot motion in shared spaces becomes a central operational challenge. Typical space-sharing conflicts can lead to collisions, congestion, and deadlocks when multiple robots attempt to access the same narrow corridor, intersection, or waypoint concurrently. Coordination strategies differ along multiple dimensions, including whether coordination is static (offline) or dynamic (online), which directly affects how conflicts are detected and resolved at runtime. [3]

From an architectural perspective, multi-robot coordination is commonly discussed in terms of centralized versus decentralized decision-making. In centralized architectures, a central control agent has global information about the environment and the robots, which can support globally optimized plans; however, such architectures can be vulnerable to failures and communication issues, and may become less effective as the team grows. Decentralized approaches can be further divided into distributed and hierarchical architectures, where the latter represents a hybrid design between fully centralized and fully distributed control. These alternatives tend to improve adaptability and robustness, although the resulting solutions are often suboptimal. [3], [4]

A large number of coordination algorithms have been proposed, each reporting advantages under specific conditions. However, the existence of many proposed solutions does not by itself constitute progress: without a systematic and reproducible evaluation framework, the relative merits of competing approaches cannot be established reliably. Current practice typically evaluates each system in isolation, using heterogeneous maps, fleet sizes, and metric definitions that preclude meaningful cross-study comparison. The limiting factor is not algorithmic variety, but the absence of evaluation infrastructure capable of exposing how those algorithms behave across the full coordination pipeline, from task allocation through path planning to execution under uncertainty, in environments that faithfully reproduce intralogistics constraints such as structural bottlenecks and dynamic replanning events. This evaluation gap is the problem addressed in this dissertation, and its precise scope is defined in the following section.

1.2 Problem Statement

Coordinating an AMR fleet in intralogistics environments requires solving two distinct but tightly coupled decision layers: Multi-Robot Task Allocation (MRTA) and Multi-Agent Path Finding (MAPF). MRTA concerns how tasks are distributed across robots and is commonly studied as a core component of multi-robot coordination and collaboration [5], [6]. After tasks are assigned, MAPF requires computing collision-free motion plans for multiple robots moving concurrently through a shared workspace [7]. In practice, MRTA outputs an assignment and often an ordering of tasks, while MAPF produces time-parameterized, collision-free trajectories that must be executed concurrently. This coupling arises because task assignment determines which robots must share which parts of the environment at which times, directly shaping the conflict structure that the path planner must resolve.

Both layers are computationally challenging in the worst case. The MRTA literature includes problem variants that are NP-hard, which motivates the use of approximate and heuristic methods in practice [6]. MAPF is also NP-hard [8], and empirical benchmarking results show that the scalability of optimal solvers degrades significantly with the number of agents and map structure: on maze-like, corridor-heavy maps ICBS solves only on the order of ten to twenty agents within practical time limits, whereas on open grids it scales to roughly a hundred [9]. Related complexity results for optimal multi-robot path planning on graphs further emphasize the computational hardness of achieving optimality at scale [8]. In contrast, industrial AMR deployments can involve fleets of several hundred robots operating concurrently in the same facility [1], creating a substantial gap between what optimal solvers can tractably address and the scale at which industrial benchmarking must operate. As a result, system performance depends strongly on design choices and parameterization, reinforcing the need for systematic evaluation under realistic workloads.

In real industrial environments, these theoretical challenges are compounded by additional constraints and uncertainty. Intralogistics layouts include narrow passages and infrastructure interactions such as doors and elevators that behave as synchronization points and can create congestion if not explicitly managed by the coordinator [10]. In real deployments, factors such as

communication delays and mechanical effects can desynchronize execution from the plan, potentially invalidating previously computed schedules and requiring online intervention to avoid conflicts and deadlocks [10]. Furthermore, warehouse logistics is typically lifelong: robots are continuously assigned new goals as soon as they complete current ones, requiring persistent planning and replanning [2].

These requirements motivate coordination systems that combine planning with active supervision, so that conflicts can be detected and resolved under real-time disturbances rather than assuming perfect execution [10]. Evaluating whether a given coordination system successfully addresses these requirements demands more than measuring path optimality in isolation: it requires assessing the full pipeline, task allocation, path planning, and supervised execution, under bottleneck-laden layouts, execution disturbances, and lifelong task streams. This dissertation designs and implements an evaluation methodology for the coordination system developed at INESC TEC, enabling reproducible, systematic benchmarking under realistic intralogistics conditions.

1.3 Benchmarking Gap

Despite the extensive body of research on multi-robot planning and coordination, comparing approaches fairly and assessing their suitability for realistic scenarios remains difficult in the absence of adequate benchmarking tools [11].

Real-world fleet coordination relies on a complex, multi-layer autonomy pipeline that spans task assignment, multi-robot planning and coordination, navigation, and execution under disturbances, yet existing benchmarks often evaluate only isolated parts of this pipeline. In practice, several benchmarks consider only a subset of the overall pipeline for robot fleets, with most focusing primarily on path planning rather than end-to-end coordination [11].

Moreover, discrepancies in assumptions and environment abstractions limit the applicability of current benchmarks to realistic intralogistics deployments. Widely used grid-based MAPF benchmarks such as MovingAI rely on 2D occupancy-grid maps and assume perfect knowledge of the world, which fails to capture execution effects and sensing uncertainty that arise in realistic operations. Even intralogistics-focused benchmarks may still represent the world as a 2D occupancy grid and use simplified robot motion models, overlooking higher-level control aspects such as collision avoidance [11].

These limitations are also visible in benchmarking tools that deliberately focus on specific layers. Bench-MR provides a framework for benchmarking sampling-based motion planning for nonholonomic wheeled mobile robots, including scenarios and metrics for comparing motion-planning system components against the state of the art [12]. PathBench targets benchmarking for classical and learned path planning and emphasizes that few attempts have been made to benchmark such algorithms holistically or unify their interface [13]. While these efforts are valuable, they reinforce that many benchmarks remain component-oriented rather than end-to-end fleet-coordination evaluations. Differences in evaluation scope, scenario fidelity, and execution

assumptions make it difficult to derive consistent evidence about the end-to-end impact of coordination strategies on global fleet performance, particularly when task assignment, path planning, and runtime disturbances interact [11].

This study defines a systematic and reproducible benchmarking methodology tailored to fleet coordination in realistic intralogistics settings. Simulation-based frameworks such as *Patrolling_Sim* illustrate how benchmarking and simulation can support running and comparing algorithms across scenarios and team sizes while enabling fair comparisons through repeatability [14]. However, *Patrolling_Sim* targets persistent area surveillance workloads, in which robots are assigned to patrol predefined routes or regions, rather than the lifelong point-to-point transport workloads characteristic of intralogistics, where throughput, bottleneck contention, and the coupling between task allocation and path execution are the primary performance drivers.

1.4 Objectives and Scope

The benchmarking gap identified in Section 1.3 motivates the need for a systematic and reproducible way to evaluate end-to-end fleet coordination in realistic intralogistics environments. This dissertation therefore designs and implements a benchmarking methodology and the corresponding experimental infrastructure to evaluate the INESC TEC fleet coordination system, which relies on the TEA* planner, a graph-based coordinator with a temporal dimension supporting wait and give-way actions, and a supervisor module that monitors execution and triggers replanning under real-world disturbances [10].

Concretely, the expected contributions are:

- **Automatable benchmarking pipeline:** Development of software to automatically execute large batches of coordination experiments within a ROS-based setup, improving repeatability and reducing manual effort.
- **Standardized methodology:** Definition of an experimental protocol with scenarios representative of intralogistics bottlenecks, adopting clear KPIs for each module:
 - **Scheduling KPIs:** Total planning time, Solution cost, Real execution time.
 - **Trajectory planning KPIs:** Total planning time, Solution cost, Number of failed plans, Number of replans, Real execution time.
- **Realism-oriented evaluation setup:** Construction of test environments that stress the coordinator primarily through structural contention and workload variation, consistent with the reference system abstraction.
- **Baseline implementation for comparison:** Implementation of a semaphore-based traffic-management baseline approach, to enable controlled comparisons against the INESC TEC coordinator.

- **Scalability and sensitivity study:** Evaluation of performance trends as fleet size and workload intensity increase, producing comparative curves across methods.

The scope is centered on centralized coordination for indoor, ground-based AMR fleets in intralogistics-like settings. In line with the reference system design, low-level control and localization remain robot-side responsibilities; the benchmark targets fleet-level planning, supervision, and coordination behavior rather than hardware development or controller tuning [10]. In this work, the execution-side coordination pressure is induced structurally, through bottleneck contention and workload density, rather than through explicitly injected disturbances; the injection of stochastic delays and communication faults is left as future work.

1.5 Dissertation Structure

The remainder of this dissertation is organised into seven chapters. Chapter 2 introduces the background and fundamental concepts required to frame the work, describing the layered fleet coordination architecture, the formal MAPF definitions, the main coordination paradigms, and the INESC TEC reference system. Chapter 3 reviews the state of the art on real-world intralogistics coordination, task allocation, MAPF planning, execution supervision, and existing benchmarking frameworks, establishing the evaluation gap that this work addresses.

Chapter 4 defines the benchmarking methodology, specifying the experimental protocol, the independent, dependent, and controlled variables, the test scenarios, and the scientific hypotheses under evaluation. Chapter 5 describes the system implementation, covering the automated ROS benchmarking pipeline, the node analyzer, the semaphore baseline coordinator, and the analysis tool.

Chapter 6 presents the experimental results obtained for both coordinators across the two scenarios, reporting the scalability, plan fidelity, replanning, and safety KPIs. Chapter 7 interprets these results against the formulated hypotheses and discusses the methodological contributions and limitations of the study. Finally, Chapter 8 summarises the main conclusions and outlines directions for future work.

Chapter 2

Background and Fundamental Aspects

This chapter establishes the foundational concepts of multi-robot fleet coordination, Multi-Agent Path Finding definitions, and coordination paradigms before detailing the INESC TEC reference system.

2.1 Fleet Coordination Architecture

Coordinating an AMR fleet in intralogistics can be framed as an autonomy stack that connects decision-making to execution: task allocation/scheduling, path planning, and supervision under disturbances. This decomposition is motivated by large-scale warehouse deployments, where robots transport inventory pods to stations and fleet-level coordination must prevent congestion and deadlocks while maintaining throughput [1], [10].

At the top layer, MRTA addresses how tasks are distributed among robots to achieve global objectives. Formal and survey work treats MRTA as a central coordination problem, and emphasizes that MRTA spans multiple variants, from simple assignment of independent tasks to time-extended scheduling of complex, constrained, multi-step tasks [6], [15]. In practice, the output of this layer is typically an assignment of tasks per robot, determining which robot executes which task and when.

Given these task-level decisions, the planning layer computes conflict-free motion for robots sharing the same workspace. In the standard MAPF formulation, the goal is to find paths for multiple agents that can be executed concurrently without collisions. Common objective functions used to evaluate multi-agent solutions include makespan and sum of costs/flowtime, which makes this layer directly measurable for benchmarking [9]. In realistic fleet systems, the output must be executable as coordinated trajectories rather than independent shortest paths, which requires coordination strategies such as prioritized planning to generate preplanned, mutually consistent trajectories [16]. Among the family of graph-based, time-extended planners, TEA* encodes explicit temporal actions on graph edges, classifying it as a centralized, online planner that continuously recalculates paths as execution feedback is received [10], [17]. For intralogistics environments, planning is often grounded on a graph representation where edges correspond to feasible actions,

including motion along a trajectory, waiting actions to resolve conflicts temporally, and interactions with shared infrastructure such as elevators [10].

The centralised architecture adopted by TEA* concentrates these layers in a single coordination module, which simplifies global consistency at the cost of scalability [18]. A supervision layer is required because execution rarely matches the plan exactly. Robust MAPF formulations address this proactively by computing k -robust plans that tolerate a bounded number of delays per agent [19]; however, real disturbances often exceed the assumed delay bound, necessitating an online supervisor. The INESC TEC coordination system explicitly addresses delays, deviations between planned paths and real execution, and communication latency to mitigate deadlocks [10]. Concretely, an Agent Supervisor module monitors feedback from robots to verify execution and, when necessary, triggers corrective actions such as replanning to preserve safety and progress [10].

The formal underpinnings of this planning layer are introduced in the following section.

2.2 MAPF Definitions: Conflicts, Objectives, and Temporal Variants

Multi-Agent Path Finding (MAPF) addresses the problem of generating non-conflicting paths for k agents moving on a shared graph $G(V, E)$, so that all agents can reach their respective goals while executing their plans concurrently [7], [9]. A MAPF solution is valid only if it avoids conflicts between any pair of single-agent plans [9].

In classical MAPF, conflicts are defined over agent positions in discrete time. The most common cases include vertex conflicts (two agents occupy the same vertex at the same time) and swapping conflicts (two agents swap locations in a single time step). Beyond these, the literature also considers additional conflict definitions such as edge conflicts (agents traverse the same edge at the same time in the same direction), following conflicts (an agent moves into a vertex occupied by another agent in the previous time step), and cycle conflicts (a set of agents rotates positions along a cycle in the same time step) [9] (see Figure 2.1).

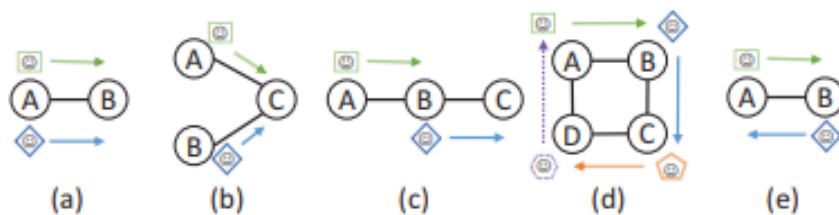


Figure 2.1: Illustration of common MAPF conflict types (edge, vertex, following, cycle, swapping). Source: [9].

MAPF solvers are typically compared using global objective functions that summarize fleet-level performance. Two widely used objectives are sum-of-costs (often associated with flowtime) and makespan (the time until the last agent reaches its destination) [7], [9]. For intralogistics fleets, makespan and flowtime are particularly informative, as congestion and waiting directly affect mission completion times [9].

In intralogistics, MAPF often appears in a lifelong form: once an agent completes a task, it is immediately assigned a new goal, producing a continuous stream of navigation problems rather than a single one-shot instance [2]. Finally, time-extended variants make the temporal dimension explicit and help resolve conflicts by reasoning about when each agent occupies critical vertices/edges, enabling actions such as wait (and, in time-aware coordinators, actions like give-way) to manage bottlenecks without purely spatial detours [10], [17].

2.3 Coordination Paradigms

Multi-robot fleet coordination can be classified by the decision-making architecture that determines where information is aggregated and where decisions are produced. A common taxonomy distinguishes centralized from decentralized architectures, the latter further subdivided into distributed and hierarchical (hybrid) designs, each inducing different trade-offs in optimality, scalability, robustness, and communication requirements [3] (see Figure 2.2). These trade-offs are particularly relevant in intralogistics, where congestion and deadlocks emerge around shared resources and coordination must remain reliable under execution delays and communication constraints.

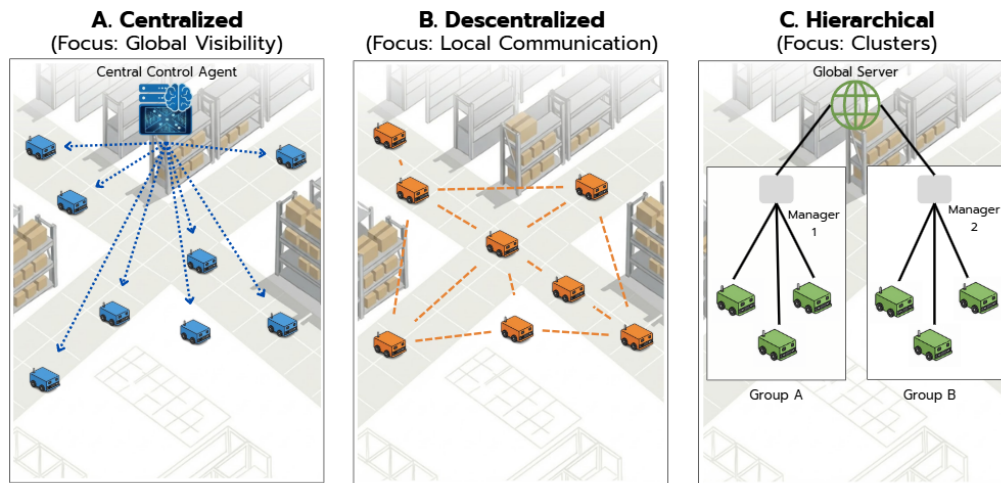


Figure 2.2: Conceptual illustration of fleet coordination architectures.

In a centralized architecture, a central control agent has global information about the environment and the robots and can communicate with the team to coordinate actions. This global visibility enables the system to compute globally consistent plans and to explicitly handle complex interactions that can lead to congestion or deadlocks in cluttered layouts [3], [10]. Centralized coordination is reported as a practical approach for industrial intralogistics applications, as global knowledge simplifies conflict resolution and enables deterministic scheduling at fleet scale [1], [18]. However, centralized systems are vulnerable to failures at the coordinator and can become less effective for large teams due to computational and communication bottlenecks, they are also sensitive to uncertainties such as communication failures or dynamic changes [3], [10].

In decentralized architectures there is no central control agent: robots are equal with respect to control and make decisions autonomously, typically using local sensing and limited communication/negotiation [3]. This tends to improve reliability, flexibility, adaptability, and robustness to unknown or changing environments when compared to centralized control [3]. In exchange, solutions are often suboptimal because decisions are made with partial information and local priorities [3]. A concrete example is decentralized waypoint-based coordination: when a robot requests a waypoint that is currently occupied, it can treat that waypoint as an obstacle and generate an alternative waypoint to reduce waiting caused by waypoint mutual exclusion, if no alternative can be generated, the robot waits until the waypoint is released [4].

Hierarchical architectures introduce one or more local coordinating agents that organize robots into clusters, forming an intermediate design between centralized and decentralized control [3]. In practice, hierarchical designs are often motivated by the need to reduce centralized bottlenecks while still exploiting some level of global structure. As a result, hierarchical approaches can improve scalability and responsiveness relative to a single global coordinator, but they may still produce suboptimal outcomes compared to fully centralized optimization and can inherit complexity from both levels of control [3].

2.4 Reference System

This research is grounded on the centralized, time-aware fleet coordination system developed at INESC TEC, which serves as the reference coordinator for the benchmarking methodology. The system follows a centralized architecture in which a central coordination module computes and dispatches fleet-level plans, while each robot retains local autonomy for localization and low-level control [10].

The coordinator represents the operating environment as a graph whose vertices encode decision points and whose edges encode actions that a robot can execute. Importantly, edges are not limited to “move from A to B” primitives: they can represent infrastructure interactions such as calling an elevator or opening an automatic door, enabling the planner to reason about shared resources and synchronization within the same model [10].

During system setup, the operating environment is first mapped and subsequently converted into a graph representation (see Fig. 2.3). This logic is further detailed in Fig. 2.4, where the V notation represents vertices (states) and E denotes edges (actions). In this notation, each vertex is spatially defined by associated (X, Y) coordinates. Crucially, each edge is associated with an execution cost defined in time, representing the duration required to perform the corresponding action rather than the distance traveled.



Figure 2.3: Example of an environment graph used for fleet coordination. Source: [10].

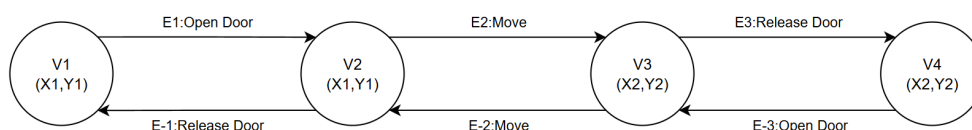


Figure 2.4: Example of encoding an infrastructure interaction/action in the graph notation. Adapted from [10].

At fleet scale, decision making is performed by a Central Coordination Module composed of four key submodules: a Graph Server, a Graph Pre-Processor, the Path Planner (TEA*), and an Agent Supervisor. The Graph Server distributes the graph used by all agents and estimates the temporal cost of each edge, including travel edges and non-travel actions. Using time as the planning unit supports a unified treatment of motion and infrastructure actions [10]. The Graph Pre-Processor analyzes the graph to identify parallel edges that interfere due to robot footprint overlap and choke points that can create flow problems, choke points can be blocked in a semaphore-like manner to avoid unsafe simultaneous use [10].

The path planner is TEA*, which extends A*-style graph search with an explicit temporal dimension so that each graph element can be reasoned about in space-time [10], [17]. This temporal reasoning enables coordination behaviors that are central to intralogistics bottlenecks: wait conditions, where one or more agents pause to let others clear critical regions, and give-way conditions, where an agent temporarily moves to a less advantageous position to allow another to pass, then resumes [10]. TEA* is also described as an online method: when new robot position readings arrive, paths can be recalculated to cope with execution delays and deviations, contrasting with purely offline planning [17].

Because real execution is affected by disturbances (e.g., unexpected obstacles, communication delays, slippage/mechanical delays), the system includes an Agent Supervisor module that monitors execution and decides when replanning is necessary. The supervisor assesses interactions between agents and can classify pairs as "Danger Close" when footprint-level proximity indicates reduced safety margins, replanning decisions are driven by the step difference between agents (Delta_steps), with stricter thresholds for "Danger Close" pairs than for "Safe" pairs [10]. At a

systems level, this supervision logic supports online intervention without replanning continuously, balancing robustness against computational and behavioral side effects [10].

Unlike decentralized execution policies such as the Minimal Communication Policy (MCP) [19], in which each agent independently decides whether to advance or wait based on the original planned visit order at each vertex, requiring no central supervisor and minimal inter-agent communication, the Delta_steps approach retains global fleet awareness to trigger explicit replanning under arbitrary disturbances. While MCP achieves robustness through local, order-preserving rules, it offers no mechanism to recover plan optimality when execution deviates significantly, making reactive centralized supervision necessary in environments where bottleneck contention can amplify small delays into deadlocks [10].

This reference system makes benchmarking particularly meaningful because it exposes measurable, design-dependent behaviors across the autonomy stack: graph modeling choices (actions and temporal costs), choke-point handling, time-aware coordination actions, and supervision-driven replanning under disturbances. The benchmark targets the two main modules of fleet management (scheduling and trajectory planning) and evaluates them with module-specific KPIs. Fleet-level indicators are also logged to capture intralogistics bottleneck effects and translate technical behavior into operational evidence.

2.5 Summary

This chapter established the fundamental concepts required to benchmark fleet coordination. It defined the layered autonomy stack, classified coordination architectures, and detailed the IN-ESC TEC reference system.

Understanding this theoretical baseline provides the necessary context for the next chapter, which will analyze the State of the Art to reveal the inability of current tools to evaluate such integrated dynamics, thereby justifying the need for the proposed benchmarking methodology.

Chapter 3

State of the Art

Fleet coordination in intralogistics has progressed from largely static automation to large deployments of AMRs operating in shared indoor spaces. At industrial scale, coordination is not only about finding collision-free paths, but also about sustaining high throughput while avoiding congestion and deadlocks under continuous operation.

3.1 Real-world fleet coordination in intralogistics and warehouses

A canonical example is the Kiva Systems, in which hundreds of robots move inventory pods to stationary picking stations. In this setting, fleet-level coordination is performed by a centralized job management layer that assigns work and routes robots using heuristics that combine travel distance and inventory availability [1]. The reported system scale (hundreds of vehicles in a single facility) illustrates why coordination becomes a primary operational bottleneck: small local interactions can cascade into global congestion if not explicitly managed. In terms of operational reporting, early large-scale deployments emphasize throughput-oriented measures as high-level evidence of efficiency [1]. Modern deployments further reinforce that coordination must be treated as a continuous (lifelong) problem. In warehouse-like workloads, robots are repeatedly reassigned new goal locations immediately after finishing a task, which requires a persistent flow of conflict-free trajectories rather than solving a single one-shot instance [2], [9]. This matters when comparing approaches: in lifelong settings, performance depends not only on single-instance solution quality, but also on how gracefully the system handles repeated reassignment, dense traffic, and accumulated delays over time.

A second practical dimension concerns environment heterogeneity and interoperability. Contemporary factory settings typically operate fleets of 50 to 200 AMRs of different types and vendors, sharing the environment with human workers, forklifts, and legacy AGVs that cannot be controlled by the fleet management system. This heterogeneity introduces coordination constraints that extend beyond pure robot-to-robot conflict resolution: AMRs must yield to faster human-driven vehicles, avoid blocking lanes used for manual loading, and operate safely near personnel. To address vendor fragmentation, industry-driven standards such as VDA5050 have

been developed to define a common communication interface between a central fleet manager and AMRs from different manufacturers, enabling a single coordination system to manage a mixed fleet. The open-source framework OpenRMF builds on similar principles, providing fleet management and multi-agent planning tools designed for interoperable robot navigation [20]. At the architectural level, the industrial decision-making stack is a layered hierarchy spanning low-level control through task assignment, with cycle times ranging from tens of milliseconds at the safety layer to tens of seconds at the planning layer, reinforcing that real-time coordination constraints propagate across the entire stack [20].

A further practical shift observed across real-world systems is the modeling of the environment and shared resources. Kiva-style deployments are typically described using a 2D grid abstraction aligned with storage cells and station locations [1]. In contrast, more recent industrial coordination frameworks adopt graph-based representations in which edges encode actions rather than only move primitives as explained in section 2.4. This richer representation supports explicit handling of structural bottlenecks (choke points) that behave like semaphores, where mutual exclusion must be orchestrated to prevent deadlocks and gridlock in narrow passages or resource entrances [10].

Finally, real deployments highlight that execution uncertainty is unavoidable and must be handled at runtime. The INESC TEC system addresses this with an explicit supervision layer that monitors execution and triggers replanning when deviations exceed a configured tolerance, aiming to prevent near-collisions and deadlocks before they fully materialize [10].

Across these deployments, a common trend emerges: coordination is moving towards lifelong planning on semantic graphs that explicitly model infrastructure constraints. While early grid-based systems maximized throughput in highly controlled environments, modern applications require coordinators capable of managing the friction of real-world disturbances without entering deadlock states.

3.2 Task allocation and scheduling in robot fleets

In intralogistics fleets, MRTA is the decision layer that continuously selects who does what, and when, under capacity, precedence and resource constraints.

3.2.1 Families of approaches

A practical distinction in fleet MRTA is between assignment and scheduling. Assignment addresses one-shot pairing of robots and independent tasks, often formalized as an Optimal Assignment Problem (OAP) [6]. This abstraction is attractive because it admits efficient solvers (e.g., Hungarian method), but it becomes brittle when logistics decisions require temporal structure (sequencing, precedence, waiting, deadlines) or when travel times depend on other agents' actions [5], [6]. Scheduling captures these time-extended variants by jointly deciding task allocation and ordering over time, taxonomies highlight that these variants are typically NP-hard, reflecting the combinatorial growth of feasible schedules once temporal coupling is introduced [6], [15].

A second axis that directly matters for the benchmark is whether tasks are static or dynamic. Static formulations assume all tasks are known a priori, enabling global optimization but often limiting scalability and realism for operations where demand evolves continuously. In contrast, dynamic formulations motivate policies that involve the real-time assignment of tasks to robots [5].

Objectives differ mainly because industry and academia answer different questions. Industrial fleet reports usually emphasize throughput because the goal is to keep the operation productive over long periods. Academic MRTA studies often optimize makespan or flowtime because these objectives are easy to define and compare across algorithms in controlled experiments [15]. However, in congestion-heavy warehouses, a schedule with a good makespan and flowtime can still execute poorly if many robots are sent through the same bottlenecks, creating queues and waiting that the objective does not capture. For this reason, fleet-oriented studies increasingly report utilization and workload balance as complementary indicators: they help reveal whether the policy is overloading a subset of robots and implicitly building queues around busy stations or shared resources [5].

A third structural axis concerns the coordination mechanism used to reach an assignment. Centralised optimisation approaches require global state visibility and scale poorly with team size. By contrast, market-based approaches use decentralised auctions in which robots bid for tasks based on local cost estimates, and an auctioneer selects the winning allocation [21]. Auction-based methods are attractive because they are modular, anytime, and naturally handle dynamic task arrivals; their main limitation is that sequential single-item auctions can produce suboptimal bundles when tasks are complementary, motivating combinatorial and multi-round variants at higher computational cost [6], [21].

3.2.2 Coupling with path planning and coordination

A recurring theme in practice is the choice between layered and integrated pipelines. Layered designs decouple scheduling from motion planning to preserve modularity and simplify interfaces; Kiva-style architectures exemplify this separation by assigning work at the job-management layer while leaving local navigation to individual agents [1]. Integrated approaches instead solve allocation and collision-free planning jointly, which can yield globally consistent decisions but quickly becomes computationally demanding as fleet size and task streams grow [11], [15].

In intralogistics, tighter coupling between scheduling and path coordination becomes valuable mainly when contention is high. If the layout contains bottlenecks such as narrow corridors, merges, doors, or elevators, the travel time of robot r to execute task t is no longer determined only by geometry: it also depends on when other robots are expected to traverse the same constrained resources, creating cross-schedule dependencies [10], [15]. Under these conditions, distance-only utility models (Euclidean/Manhattan) tend to be overly optimistic because they ignore queueing and mutual exclusion, and therefore underestimate effective execution time [6], [11]. The consequence is not just a slightly worse schedule: a set of individually sensible assignments can collectively push too many robots into the same choke points, increasing waiting, propagating

delays, and in extreme cases producing deadlock-like gridlock when contention is not reflected in the scheduling cost [4], [18]. Crucially, this failure mode motivates specific benchmark scenarios: simplified, layered scheduling is expected to degrade under bottleneck pressure, whereas coordination-aware costs should yield measurable improvements.

3.2.3 Evaluation practice and missing pieces

MRTA evaluation is still dominated by algorithmic metrics: runtime to produce a schedule, objective value (often makespan/flowtime), and utilization measures under simplified dynamics [5]. However, these metrics often assume deterministic traversal times and perfect state information, implicitly favouring low-slack schedules that may not survive execution noise and delayed updates [1], [6]. This gap becomes visible when compared to high-TRL coordination reports, which describe execution deviations and latency as routine phenomena, and where supervisory interventions are a necessary mechanism to maintain progress under congestion and shared-resource contention [10], [16].

Two omissions are particularly consequential for benchmarking. First, schedule stability is rarely quantified: reassignments and preemptions may improve feasibility but can introduce operational oscillation and throughput loss if they occur too frequently [1], [5]. Second, fairness is often treated as secondary, despite its impact on long-horizon operation and on the emergence of persistent bottlenecks [5], [18]. For a fleet benchmark, these gaps are not cosmetic: they determine whether an MRTA approach remains reliable as contention, disturbances and team size increase.

3.3 MAPF and planning methods for coordination

This section reviews the MAPF literature, focusing on the transition from abstract solvers to deployable coordination strategies. The analysis is structured into three parts: a taxonomy of solver families, the specific trade-offs required for intralogistics environments, and a critique of current evaluation practices, highlighting the gap between nominal planning metrics and operational reality.

3.3.1 Solver families and coordination paradigms

Optimal MAPF research is largely structured around search-based solvers that explicitly resolve inter-agent conflicts. Conflict-Based Search (CBS) and improved variants such as ICBS provide completeness and optimality by iteratively identifying collisions and resolving them by imposing specific spatiotemporal constraints (e.g., forbidding Agent A from using location x at time t), effectively growing a high-level search tree of conflict-free variations [7], [22]. In fleet-relevant regimes, these guarantees come with a practical limitation: runtime can degrade sharply in instances where conflicts proliferate, making them mostly suitable as baselines, small-team solvers, or as components inside decomposition mechanisms such as Independence Detection [7], [23].

At scale, many systems adopt bounded-suboptimal or heuristic coordination paradigms that trade guarantees for throughput. Prioritized Planning (PP) computes paths sequentially based on a pre-defined ordering, where each agent treats the trajectories of higher-priority predecessors as dynamic obstacles. This approach achieves low runtimes but loses completeness and often suffers in constrained layouts, its practical success is tightly linked to structural assumptions (e.g., well-formed infrastructures where waiting at endpoints does not block traffic) [16]. Priority-Based Search (PBS) partially mitigates the brittleness of fixed priority orders by dynamically determining which agent goes first only when a specific conflict is detected, thereby searching the tree of possible priority relations rather than adhering to a single global sequence [2], [24]. For intralogistics, the key point is that these families represent two distinct conflict resolution strategies: either enforce conflicts away explicitly with expensive global search, or accept a controlled level of suboptimality to obtain responsiveness under load.

A second axis that matters operationally is whether the planner reasons only in space, or in space–time. In shop-floor coordination, producing merely geometric paths is insufficient: plans must encode when agents occupy shared regions so that wait, yield, and mutual exclusion can be expressed directly. Time-extended planners address this by adding time as an explicit planning dimension, turning conflicts into space–time constraints and enabling deliberate coordination actions [17]. Within this family, TEA* is representative of an online, time-aware approach: it maintains temporally indexed occupancy while continuously refreshing plans as new state readings arrive, which makes it particularly aligned with lifelong operation and disturbance-prone execution [10], [17].

A landmark framework for lifelong MAPF in warehouse-scale environments is Rolling-Horizon Collision Resolution (RHCR) [2]. RHCR decouples the continuous task-assignment problem from conflict resolution by defining a bounded time window w : at each planning cycle, an underlying solver (e.g., PBS) is invoked only within that horizon, and the window slides forward after each execution step. This design limits the conflict search to interactions relevant for near-term execution, improving scalability to hundreds of agents on large grid maps. The cost is the absence of global completeness guarantees; conflicts outside the active window are deferred to future cycles and may be resolved suboptimally. Experimental results in [2] demonstrate that throughput saturates beyond a relatively small window size, suggesting that short-horizon replanning is sufficient for steady-state warehouse operation. In contrast, TEA* does not commit to a fixed time horizon: occupancy is maintained as a running temporal index refreshed reactively in response to execution state, and the replanning trigger is governed by the supervision layer’s `Delta_steps` threshold rather than a periodic window boundary [10], [17]. This distinction is consequential for benchmarking: in RHCR, the window parameter w directly controls the latency-throughput trade-off; in TEA*, the analogous lever is the supervisor sensitivity threshold, which operates at execution time rather than at planning time.

3.3.2 Key trade-offs for intralogistics coordination

Across solver families, the dominant trade-off is optimality vs. scalability. CBS-like approaches can produce globally optimal coordinated plans but typically become impractical as fleet size and interaction density rise, whereas priority-based and rolling-horizon heuristics scale to large fleets but sacrifice completeness and can accumulate inefficiencies under contention [2], [7].

A second, intralogistics-specific tension is robustness vs. replanning overhead. Plans that are tightly synchronized can be fragile: small execution delays may propagate, forcing frequent global repairs and increasing systemic delay. Robust planning variants introduce temporal slack to absorb delays, but pay for this stability with higher planned cost and potentially lower nominal throughput [19]. This trade-off is not purely algorithmic: it is shaped by the supervision policy and by how aggressively the system triggers replanning when desynchronization is detected [10].

Finally, bottlenecks and shared-resource interactions determine which coordination mechanism is viable. In layouts dominated by choke points (narrow corridors, merge points, doors, elevators), purely spatial detours may not exist, and time-slotting becomes the only resolution mechanism. This makes space–time planners and reservation-based coordination particularly relevant for intralogistics, because they can encode mutual exclusion through planned waiting instead of relying on geometric rerouting [10], [16].

3.3.3 Evaluation practice and gaps (planning vs. execution)

Standard MAPF evaluation commonly reports planning-centric KPIs, runtime, success rate, and nominal cost, often measured on abstract 2D grid instances [9], [23]. While this is useful for controlled algorithmic comparison, it under-represents execution-side effects that dominate fleet performance: communication latency, controller recovery behaviours, and the costs of online plan modifications [10], [19].

In real fleets, good plan cost does not necessarily imply good operational performance. This dissociation has been empirically documented across different robotic platforms. In a warehouse automation setting where centrally computed MAPF schedules are executed on physical robots, timing discrepancies accumulate across agents during execution and cause plan failures not predictable from nominal cost alone, because the executed action durations deviate from the synchronous timing the plan assumes [25]. Plan quality and execution fidelity thus decouple as fleet size and inter-agent synchronisation requirements grow. Highly synchronised solutions are therefore operationally brittle: a single delayed agent may trigger cascaded waiting, supervisor interventions, and global replans, shifting the dominant cost from nominal path length to recovery overhead and stalled time [10], [19].

3.4 Execution uncertainty, supervision, and online replanning

While section 3.3 focused on coordination at planning time, real deployments must also ensure that plans remain safe and productive during execution. Recent work explicitly highlights the gap

between nominal MAPF plans and physical execution, where delays and imperfect state information can invalidate tightly synchronized trajectories and amplify contention at shared resources [10], [19]. This section reviews how the literature models such disturbances, how supervision/monitoring is implemented, and how online recovery is performed and evaluated.

3.4.1 Disturbance models

A large portion of the MAPF literature still evaluates planners under deterministic execution assumptions, primarily to isolate algorithmic properties and enable controlled comparisons [9], [16]. However, fleet-oriented work increasingly introduces explicit disturbance models. The most common abstractions are timing uncertainty (e.g., slippage or actuation delays), noisy perception/state estimation, and communication imperfections [10], [19].

In discrete-time MAPF, timing uncertainty is often modeled as stochastic delays where an agent fails to execute a planned move with probability p , effectively inserting unplanned "wait" actions into the execution trace [19]. Although simplistic, this model captures the main coordination consequence in dense infrastructures: delay propagation. When agents are tightly coupled through narrow corridors, doors, or elevators, a single delayed agent can violate the assumed temporal separation that kept trajectories collision-free, forcing downstream interventions [10], [19]. Communication is similarly idealized in many coordination pipelines, but robust-execution work relaxes this assumption by considering delayed synchronization (e.g., agents can communicate or sync only every T steps) or partial dropout, which increases the risk of stale reservations and late conflict detection [10], [19].

A key limitation across these models is what they omit: disturbances are frequently treated as independent, step-wise events, and continuous-time tracking error, localization drift, and dynamic obstacles are either abstracted away or handled outside the MAPF layer [10], [16]. For benchmarking intralogistics stacks, these models clarify why supervision is required and how delays/stale information can propagate. In this dissertation, the experimental factors are induced primarily through scenario bottlenecks/shared resources and workload variation, which are sufficient to expose waiting, interventions, and replanning overheads in end-to-end runs.

3.4.2 Supervision and monitoring mechanisms

To bridge planning and execution, the literature introduces a supervision layer that monitors deviations and decides when corrective action is warranted. Three paradigms recur across the literature, and their properties differ substantially in terms of formal guarantees, replanning requirements, and computational overhead.

The Minimal Communication Policy (MCP) is an execution policy that preserves the planned vertex-visit order as an invariant: before agent i enters vertex v , MCP verifies whether i is the next agent scheduled to enter v according to the original plan; if not, i waits until the preceding agent has departed [19]. This ordering invariant eliminates the need for runtime replanning and has been formally proven to be a robust-execution policy with respect to any plan that is k -robust for

$k \geq 1$ [19]. A plan is k -robust if and only if it contains no k -delay conflicts, meaning it tolerates any combination of per-agent delays not exceeding k steps without collisions [19]. The cost overhead of k -robust plans is modest for small k (below 5% for $k=1$ on open grids), but IkR-CBS planning runtime grows sharply: from 5 ms at $k=0$ to over 9,200 ms at $k=7$ on a 16×16 grid with five agents [19]. Critically, MCP requires a base plan that is at least 1-robust (MAPF-DP). Separately, lazy repair variants have been shown to produce deadlocks in scenarios where forcing either agent to wait cannot resolve the conflict [19].

A complementary perspective on supervision guarantees is provided in [16], where classical Prioritized Planning is shown to be incomplete: a robot at its goal position can permanently block a lower-priority robot from reaching its destination. Their Revised Prioritized Planning (RPP) recovers completeness under the condition that trajectories avoid the start and goal regions of higher-priority robots [16]. In environments dominated by choke points, these structural conditions frequently do not hold, making supervision and runtime recovery architecturally necessary rather than optional.

In the TEA* system, execution supervision is implemented through a Δ_{steps} indicator, defined as the step difference between any pair of agents relative to their planned trajectories [10]. Rather than applying a uniform global threshold, the supervisor classifies each agent pair as *Danger Close* (in close spatial proximity within a configurable time window) or *Safe* (relatively isolated), applying distinct thresholds respectively. This context-aware discrimination avoids unnecessary replanning when agents are spatially separated while ensuring timely intervention near shared bottleneck resources. Empirical validation across simulation and real deployments shows that all test runs were completed without deadlock [10].

Table 3.1: Comparative analysis of principal supervision mechanisms for multi-robot coordination.

Mechanism	Safety guarantees	Trigger & computational cost	Principal failure mode
MCP (with k -robust plans) [19]	Proven collision-free as an execution policy for any k -robust plan ($k \geq 1$)	No runtime replanning; high upfront planning cost (e.g., IkR-CBS grows exponentially with k)	Requires prior knowledge of delay bound k ; lazy repairs can deadlock; impractical for large fleets.
Revised Prioritized Planning (RPP) [16]	Completeness guaranteed only in well-formed infrastructures	Lower-priority agents recalculate trajectories to avoid higher-priority ones	Fails and can deadlock in choke-point-rich environments where structural conditions do not hold.
Δ_{steps} + Danger Close / Safe [10]	Practical: no deadlock or collision across all tested runs; LiDAR halt as last resort	Fires when $\Delta_{\text{steps}} > s_0$ (Danger Close) or $> s_1$ (Safe); < 1 s per replan (avg. 0.356 s in hospital env.)	No formal proof; threshold parameters must be tuned to environment topology

These paradigms occupy complementary regions of the design space. MCP is most appropriate when at least a 1-robust plan is available and the fleet is small enough for IkR-CBS to terminate within the planning budget; it is not directly applicable to TEA*, whose online planner does not produce k -robust plans in the formal sense. The completeness guarantees of RPP [16] similarly hold only in well-formed infrastructures, which choke-point-rich benchmark scenarios

are designed to violate. The Delta_steps mechanism requires no prior knowledge of k and adapts supervision sensitivity to spatial proximity, making it the natural operating mechanism for real-time deployment under unpredictable delay distributions.

The Delta_steps mechanism is also the appropriate reference for this benchmark for three specific reasons. First, it is integral to the TEA* coordination stack: the benchmark evaluates the system end-to-end, and the supervision layer is an architectural component that determines when replanning is invoked during execution. Second, unlike MCP, it operates without a prior estimate of k , which matches the realistic condition of intralogistics environments where the delay distribution is not known in advance. Third, the Danger Close classification is directly aligned with the benchmark's structural contention scenarios: bottlenecks create Danger Close regions by construction, making the supervisor selectively sensitive precisely where coordination pressure is highest.

3.4.3 Online replanning and recovery strategies

Once a violation is detected, recovery strategies range from lightweight local actions to full global replans. A common first line is local repair, which attempts to absorb disturbances by selectively inserting waits or enforcing local precedence constraints so that delayed agents can rejoin the nominal ordering (e.g., MCP-style policies) [19]. In more decentralized coordination, local repair can be framed as short-horizon rerouting around temporarily blocked waypoints, treating the occupancy as a dynamic obstacle (e.g., Alternative Waypoint Generating) [4].

However, local repair is fragile in precisely the cases that dominate intralogistics benchmarking difficulty: choke points and shared infrastructure interactions where spatial detours may not exist and mutual exclusion must be expressed temporally [10], [19]. In such regimes, supervisors may trigger global replanning, recomputing a conflict-free plan from the current fleet state [10]. Global replanning improves feasibility but introduces its own failure mode: replanning cascades, where the time to compute and disseminate a new plan approaches (or exceeds) the time horizon over which that plan remains valid under ongoing disturbances [10], [19]. This motivates hybrid schemes (rolling-horizon updates and partial replans) that aim to limit disruption while still resolving hard conflicts.

3.4.4 Evaluation practice and remaining gaps

Evaluation of execution uncertainty typically reports a limited set of outcomes: number of replans, success rate, and increases in nominal cost metrics under disturbances. While valuable, these metrics often under-describe the operational reality of intralogistics, where performance degradation is frequently dominated by congestion, mutual exclusion at shared resources, and supervision interventions triggered under high traffic density [10].

To meaningfully measure the project's core KPIs, the coordination system must be evaluated under operating regimes in which conflicts are frequent. If the testing environment lacks

contention, the measured KPIs tend to mirror nominal behavior, making it difficult to expose coordination overheads such as waiting and interventions.

Accordingly, the benchmark is designed so that coordination pressure emerges naturally from the experimental conditions, through controlled variation of scenario structure and workload density. Two factors are systematically varied: structural contention, induced by explicit bottlenecks and shared resources in the environment graph, and traffic pressure, controlled through fleet size sweeps into near-capacity regimes where supervision and replanning become frequent [10]. This design ensures that the core KPIs are measured under conditions where coordination overheads are observable rather than negligible.

3.5 Benchmarking frameworks and evaluation methodologies

This section reviews the current landscape of benchmarking tools. Existing evaluation tools can be grouped by which autonomy layer they exercise, a distinction that largely determines their scalability, realism, and reported KPIs.

3.5.1 Benchmark families

Building on the benchmarking gap discussed in Section 1.3, existing evaluation tools can be grouped by which autonomy layer they exercise rather than by the specific algorithm under test. This distinction matters because scalability, realism, and reported KPIs are largely determined by the abstraction level.

MAPF suites such as MovingAI provide large sets of grid instances and standardized cost metrics, enabling controlled solver comparisons at scale [9]. Their limitation for intralogistics benchmarking is the abstraction of robots as zero-volume grid agents with perfect execution, which prevents measuring execution-side effects and supervision overhead [9].

Benchmarks like Bench-MR evaluate non-holonomic feasibility and path quality metrics (e.g., curvature, Angle-Over-Length) for wheeled robots [12]. PathBench targets classical and learned planners under a unified interface and emphasizes planner-comparability, but still remains primarily single-robot planning-centric for industrial coordination questions [13].

Arena-bench, for instance, introduces dynamic agents (humans) and reports navigation-centric metrics (e.g., collisions, smoothness/jerk), which is valuable for social navigation but not designed to stress shared-resource contention or fleet-level scheduling effects [26].

Frameworks such as Patrolling_Sim provide ROS-based batch experimentation, enabling systematic sensitivity studies across team sizes and scenarios [14]. However, they target patrolling workloads and typically do not model intralogistics-specific resources (e.g., lifts/doors) or execution disturbances as first-class evaluation dimensions.

MRP-Bench is a recent attempt to benchmark multi-robot planning in realistic 3D simulation using ROS 2, Gazebo, RMF and Nav2, aiming to bridge algorithmic planning and deployable navigation stacks [11]. The trade-off is computational cost: higher fidelity tends to restrict scale compared to grid-based suites [11].

3.5.2 Criteria for comparative analysis of benchmarking tools

To compare frameworks consistently, the literature suggests evaluating each platform along the following methodological dimensions:

- **Environment fidelity vs. scalability:** grid abstractions scale to very large fleets but omit geometry and dynamics, 3D+Nav stacks capture execution effects but limit the number of agents per run [2], [11].
- **Workload model:** one-shot instances versus lifelong operation, which changes congestion regimes and makes replanning/supervision central [2], [11].
- **Bottlenecks and shared resources:** whether the benchmark explicitly represents mutual exclusion (doors, lifts, choke points) rather than only free-space collision constraints [10], [11].
- **Execution realism and supervision:** whether deviations, delays, or communication constraints exist and whether the platform logs supervisor interventions/replanning impact [10].
- **Automation and reproducibility:** batch execution, seeds, standardized inputs/outputs, and pipeline support for sensitivity analysis rather than single nominal runs [12], [14].

Table 3.2 synthesizes the analysis of existing frameworks against these dimensions. As shown, no single platform currently combines high scalability, lifelong workload modeling, and execution disturbances suitable for industrial fleet supervision.

Table 3.2: Coverage of representative benchmarking platforms across coordination layers and realism assumptions.

Work / Platform	Layer	Lifelong	Bottlenecks	Disturbances	Robot model	Automation	Main KPIs
MovingAI [9]	MAPF	No	No	No	Point/Grid	High	Makespan, Cost
Patrolling_sim [14]	Exec.	Yes	No	No	2.5D (Stage)	Medium	Idleness
Bench-MR [12]	Plan.	No	No	No	Non-holonomic	High	Curvature
PathBench [13]	Plan.	No	No	No	Point/Grid	High	Success Rate
Arena-bench [26]	Nav.	No	No	Humans	3D (Gazebo)	Medium	Collisions
MRP-Bench [11]	Full	No	Lifts/Doors	No	3D (Gazebo)	Medium	Time Blocked

3.5.3 Implications for the benchmark design

Table 3.2 maps each platform against the evaluation dimensions established in Section 3.5.2, but the table alone does not explain the underlying causes behind each identified gap.

The central claim that motivates the proposed benchmark, that no existing framework simultaneously provides high scalability, lifelong workload modelling, structural bottleneck contention, and supervision-loop measurement, requires a platform-level justification by exclusion, since each limitation is architectural rather than a missing feature that could be incrementally patched.

MovingAI achieves high scalability precisely by abstracting robots as zero-volume point agents with perfect, synchronous execution [9]. This is not an accidental design choice: eliminating the execution layer is what permits solver comparison at fleet sizes and map densities that would be computationally infeasible in physics-based simulation. However, the same decision structurally prevents the platform from capturing lifelong task streams, execution deviations, or supervision-triggered replanning. There is no execution layer in which timing disturbances can occur, no task generator to sustain continuous operation, and no mechanism to log supervisor interventions. Lifelong MAPF can be studied using MovingAI map files, but the benchmark infrastructure itself provides none of the runtime machinery required to measure the target KPIs without substantially rebuilding the evaluation stack.

Patrolling_Sim supports a continuous workload and provides ROS-based batch execution [14]. However, the patrolling topology contains no doors, elevators, or narrow-corridor merge points modelled as mutual exclusion constraints, the framework includes no conflict-resolution or scheduling layer. The idleness metric that the platform reports is specific to the patrolling objective and cannot be mapped to the scheduling and replanning KPIs required for intralogistics fleet evaluation.

Bench-MR and PathBench are single-robot platforms that evaluate path quality metrics, curvature, Angle-over-Length, path deviation, for individual planners [12], [13]. Neither includes a coordination layer for inter-robot conflict detection or scheduling. Fleet coordination evaluation is outside the scope of both platforms by design.

Arena-bench introduces humans as dynamic obstacles, modelled via a social force simulation, and reports navigation-centric metrics such as collision rate and jerk [26]. The platform targets single-robot social navigation and has no scheduling or task-allocation layer. Crucially, the disturbances modelled are kinematic obstacles, humans blocking the robot's path, rather than fleet-level coordination disturbances such as timing deviations between synchronized agents or supervision-triggered replanning events, which are absent from the platform.

MRP-Bench is the only existing platform that exercises MRTA, path planning, and navigation in a single stack while explicitly modelling structural bottlenecks (lifts, doors) [11]. Its main limitation is scalability: the authors explicitly acknowledge that the tool becomes computationally infeasible beyond approximately ten agents on a normal computer. Furthermore, execution-timing disturbances are not injected as a controlled experimental variable; the platform reports coordination outcomes such as Time Blocked but does not systematically test the response of a supervision layer to induced deviations.

The limitations are therefore architectural, not incidental. MovingAI's scalability depends on eliminating execution; MRP-Bench's fidelity depends on Gazebo, which caps fleet size below the regimes required to stress coordination; Patrolling_Sim's lifelong workload is coupled to a patrolling objective incompatible with intralogistics scheduling; Bench-MR, PathBench, and Arena-bench are single-agent platforms outside the fleet coordination scope. No existing platform jointly covers all four dimensions required to evaluate the TEA* coordination stack end-to-end.

To address this gap, the proposed benchmarking pipeline operates directly on the TEA* coordination stack within its native ROS environment, using a semantic graph that encodes bottleneck constraints as first-class mutual exclusion primitives. Scalability is achieved through controlled variation of fleet size and workload density on a fixed graph topology, without dependence on physics simulation. The supervision loop remains active across all runs, so replanning events and their timing cost are captured as primary measurement outputs alongside execution time and plan fidelity, the combination that no existing platform measures jointly.

3.6 Synthesis: Limitations and research gap

The review across Sections 3.1–3.5 reveals that the research landscape is fragmented along the same architectural boundaries as the coordination stack itself. MRTA evaluation is dominated by makespan, flowtime, and allocation runtime, measured under deterministic traversal assumptions that do not capture how execution time deviates from planned cost when robots contend for shared infrastructure [5], [6], [15]. MAPF solvers are benchmarked on abstract grid instances with perfect, synchronous execution; plan quality and execution fidelity are treated as equivalent by assumption, a premise that has been empirically shown to fail as fleet size and inter-agent synchronisation requirements increase [9], [25]. Execution and supervision research quantifies replanning count and recovery cost in isolation from scheduling, making it impossible to attribute replanning frequency to coordination strategy versus environment structure [19]. The benchmarking platforms surveyed in Section 3.5 inherit this fragmentation directly: each covers one layer at the expense of the others, and as established in Section 3.5.2, each limitation is architectural rather than incremental.

The gap is therefore not a missing tool but a missing evaluation unit. No existing methodology jointly measures, from a single end-to-end execution run, plan fidelity under bottleneck contention, replanning frequency as a function of workload density, planning time overhead as fleet size scales, and load balance across the fleet. These are the KPIs that determine whether a fleet coordination strategy is operationally viable in intralogistics, and they are absent as primary metrics in all three research communities reviewed [15], [19]. A coordination strategy that performs well on MRTA makespan or MAPF cost can still exhibit poor plan fidelity and high replanning frequency under shared-resource pressure, because these metrics are structurally invisible to per-layer evaluation [19], [25].

The benchmarking methodology proposed in this dissertation addresses this gap directly. The primary objective is to develop a reproducible and automatable evaluation pipeline for comparing fleet coordination strategies in intralogistics conditions. The methodology treats the TEA* stack as a single evaluation unit operating in its native ROS environment, without decoupling any layer for isolated measurement. Structural bottleneck scenarios enforce shared-resource contention at the coordination layer without dependence on physics simulation. Cross-layer KPIs spanning scheduling quality, trajectory planning cost, replanning frequency, and plan fidelity are captured

jointly from each execution run, producing the multi-layer evaluation record that Table 3.2 shows no existing framework currently provides [9], [11].

Chapter 4

Benchmarking Methodology

This chapter defines the methodology that governs the benchmarking study and that the software of Chapter 5 realises. It first specifies the experimental protocol, the independent, dependent, and controlled variables under which every run is conducted and compared. It then describes the two test scenarios, chosen to impose contrasting levels of structural coordination pressure, and finally states the five falsifiable hypotheses that the evaluation is designed to test.

4.1 Experimental Protocol

The experimental protocol defines the conditions under which all benchmark runs are conducted and establishes the rules by which coordination systems are compared.

4.1.1 Independent Variables

Two independent experimental factors were varied across the benchmarking campaigns:

- **Fleet Size (N):** The number of active robots was varied per scenario. In the industrial facility (`sim1_portic`), N took six levels, $N \in \{2, 4, 8, 12, 16, 20\}$; in the constrained hospital facility (`sim2_hosp`), N took seven finer-grained levels, $N \in \{2, 3, 4, 5, 6, 7, 8\}$.
- **Workload Configuration:** Two distinct mission sets, denoted `m1` and `m2`, were defined. `m1` represents a single-task workload (visiting a single vertex), whereas `m2` doubles the workload by assigning two sequential tasks per mission (visiting one vertex and then returning to the starting vertex), which results in higher traffic and tighter congestion within the bottleneck regions of the graph.

Crossing fleet size with workload yields 12 experimental conditions per coordinator in the industrial scenario and 14 in the hospital scenario, for a total of 26 conditions per coordinator and 52 runs across the two coordinators. To ensure a fair comparison, the exact sequence of tasks within each workload was kept identical across all evaluated coordinators, so that any performance variation is strictly attributable to coordinator behaviour.

4.1.2 Dependent Variables

The dependent variables are the KPIs logged by the benchmarking pipeline at the end of each run. Total execution time measures the duration from the moment the first robot receives its initial goal to the moment the last robot completes its final mission in the batch, capturing end-to-end fleet throughput. Per-mission execution time is the elapsed time between the first navigation command issued to a robot for a given mission and its confirmed completion. Total replanning events counts the cumulative replannings triggered for missions that had already been assigned a route, each event indicating that the original plan was invalidated during execution and a new one was computed. Planned cost is the temporal cost of the route as computed by the planner, expressed in the same units as the graph's edge weights. Failed plannings records the count of planner invocations that returned no valid plan. Finally, planning time measures the computation time required by the path planner to generate a route for a given mission.

This study injects no execution disturbances such as timing slippage or communication dropout. The desynchronisation that the supervisor resolves is instead endogenous: wait and give-way actions cause yielding robots to fall behind their planned step schedule relative to others, and the free-flow edge costs only approximate the executed trajectory. As fleet size and workload grow, more agent pairs exceed the supervisor's `Delta_steps` threshold, which is why the replanning count rises with N . This matches the behaviour reported in the reference system under its own simulations without injected noise [10].

4.1.3 Controlled Variables

The missions are seeded identically for both coordinators at each experimental condition (N , workload). This guarantees that TEA* and the semaphore coordinator receive exactly the same sequence of task requests, making the comparison of outcomes direct and unconfounded by differences in task distribution. Without this control, apparent performance differences could be artefacts of the random assignment of tasks to robots rather than consequences of coordinator design.

All runs were executed on the same machine under ROS, with identical simulator configuration, the same compiled versions of both coordinators, and the same launch parameters for all components other than the coordinator node itself.

A maximum wall-clock duration was imposed on every run. Runs are classified as `FAILED` and logged as such in the output YAML under either of two abort conditions: the accumulated planning failure count exceeds its threshold, or the run exceeds its maximum wall-clock duration before all missions complete. Deadlock is not detected directly; a deadlocked fleet stops making progress and is therefore captured by the timeout condition. Both the timeout value and the planning failure limit apply equally to both coordinators.

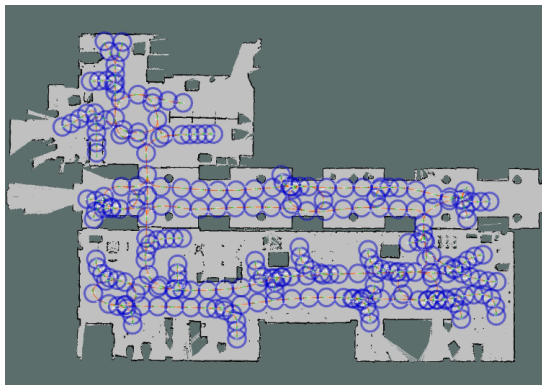
Both coordinators are evaluated through the same pipeline. The benchmarker subscribes to a standardised set of ROS topics and operates independently of both coordinators, introducing no measurement bias. The output of every run is a single YAML file.

4.2 Test Scenarios

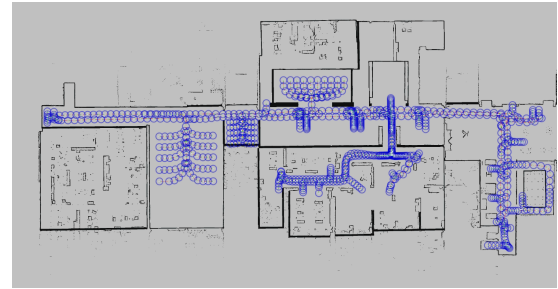
Two distinct facility environments were used as test scenarios, each modelled as a topological graph in which vertices represent waypoints and directed edges represent navigable paths with associated temporal costs. The scenarios were selected to provide contrasting levels of structural coordination pressure, enabling the benchmarking protocol to assess coordinator behaviour across environments of different complexity.

`sim1_portic` is derived from a real industrial facility and features a partially open layout organised around two main horizontal corridors with transversal connections and a denser waypoint cluster in the upper storage zone (Figure 4.1a). Bottlenecks arise at the transition points between zones, but the layout retains sufficient alternative routes to allow both coordinators to operate without systematic failure at low fleet sizes.

`sim2_hosp` is modelled on a hospital floor plan and imposes substantially greater coordination pressure (Figure 4.1b). A dense central corridor branches into narrower passages leading to individual rooms, producing a higher concentration of single-access vertices which are the topological conditions most likely to induce deadlocks under reactive coordination.



(a) `sim1_portic` — industrial facility.



(b) `sim2_hosp` — hospital facility.

Figure 4.1: Topological graphs of the two test scenarios overlaid on their respective floor plans. Blue circles represent graph nodes; edges encode navigable paths with temporal costs.

4.3 Scientific Hypotheses

The five hypotheses below follow from the coordination principles of Chapter 2 and the gaps identified in Section 3.5.

H1 — Scalability. TEA* is expected to achieve lower total execution times than the semaphore coordinator as fleet size N increases. This direction follows from the observation that centralised, plan-based fleet coordination scales to large robot teams in warehouse settings, sustaining throughput as the number of agents grows [1], [2], [10], whereas a purely reactive mutual-exclusion scheme accumulates blocking and contention at shared resources as fleet density increases.

- H2 — Bottleneck Amplification.** The performance gap between TEA* and the semaphore coordinator is larger in `sim2_hosp` than in `sim1_portic`, at equivalent fleet sizes and workloads. Realistic, complex and cluttered environments are known to amplify the difficulty of multi-robot coordination [11], the hospital scenario, being more constrained, is therefore expected to expose this gap more strongly.
- H3 — Replanning Overhead.** TEA* relies on replanning as its primary conflict resolution mechanism: when execution drifts from the planned schedule (through delays, desynchronisation, or proximity between agents), the supervisor triggers the central planner to compute a new time-extended path. The reliance on replanning as the principal response to execution-time deviations means its cost grows with the number of agents, as documented in robust MAPF execution under unexpected delays [10], [19]. The number of replanning events in TEA* grows with fleet size N , reflecting increased coordination pressure. The semaphore coordinator, which has no planning layer and resolves conflicts exclusively through reactive edge locking, produces no replanning events by design and therefore serves as a structural reference rather than a direct comparator for this hypothesis.
- H4 — Plan Fidelity.** Because TEA* computes routes using temporal edge costs that approximate real traversal times, its planned cost should be a reliable predictor of actual mission execution time. Conventional MAPF formulations assume synchronous, unit-duration actions, so the executed schedule diverges from the planned one once real dynamics are considered [25]. The plan fidelity ratio of TEA* is therefore expected to remain close to unity and stable as N increases, testing whether the temporal cost model is sufficiently accurate to avoid that divergence in the evaluated scenarios.
- H5 — Safety.** The semaphore coordinator resolves conflicts exclusively through reactive edge locking, with no mechanism to anticipate or prevent circular wait conditions. Under sufficient fleet density, groups of robots can mutually block each other indefinitely, resulting in execution-level deadlocks and failed run outcomes. TEA*, by contrast, employs a central supervisor that continuously monitors agent synchronisation and triggers replanning before conflicts escalate into deadlocks [10]. The operational success rate of the semaphore coordinator is therefore expected to degrade significantly with N , collapsing at fleet sizes where TEA* continues to complete runs successfully. This expectation rests on the premise that the space-time look-ahead of the TEA* planner prevents the circular wait conditions that the semaphore cannot resolve, while its supervisor recovers from the residual desynchronisation that arises during execution.

Chapter 5

System Implementation

This chapter describes the software developed to realise the methodology of Chapter 4. Three of its components form the measurement infrastructure: the automated benchmarking pipeline that launches and terminates every run (Section 5.1), the `mission_benchmark` node that derives the KPIs from the coordinator’s ROS topics (Section 5.2), and the analysis tool that aggregates the resulting records into comparative views (Section 5.4). The fourth, the semaphore baseline (Section 5.3), is the reactive reference coordinator against which the time-aware system is compared.

5.1 Automated ROS Benchmarking Pipeline

Conducting a systematic benchmarking campaign across multiple coordinators, scenarios, fleet sizes, and workload configurations requires that each run be launched, monitored, and terminated under identical conditions. The pipeline described in this section automates the full lifecycle of a benchmarking run, from system launch to YAML export, ensuring that every data point in the results chapter was collected under the same controlled procedure.

5.1.1 Pipeline Architecture

The pipeline is composed of three layers that operate in sequence during each run, as illustrated in Figure 5.1.

The orchestration layer is implemented as a Bash script (`benchmark_runner.sh`) that parameterises and coordinates the execution of all other components. It receives the coordinator under test (TEA* or semaphore) as a command-line argument and iterates over a predefined list of test configurations. Each configuration is specified by a mission file that defines the robots involved and the sequence of vertices each robot must visit. For each coordination system, the script launches the appropriate `roslaunch` file and waits for the fleet to initialise before triggering execution.

The measurement layer consists of two components that operate concurrently during execution. The `mission_benchmark` is a ROS node (C++) that subscribes to the coordination system’s topics and publishes derived KPIs, as described in Section 5.2. These same topics also

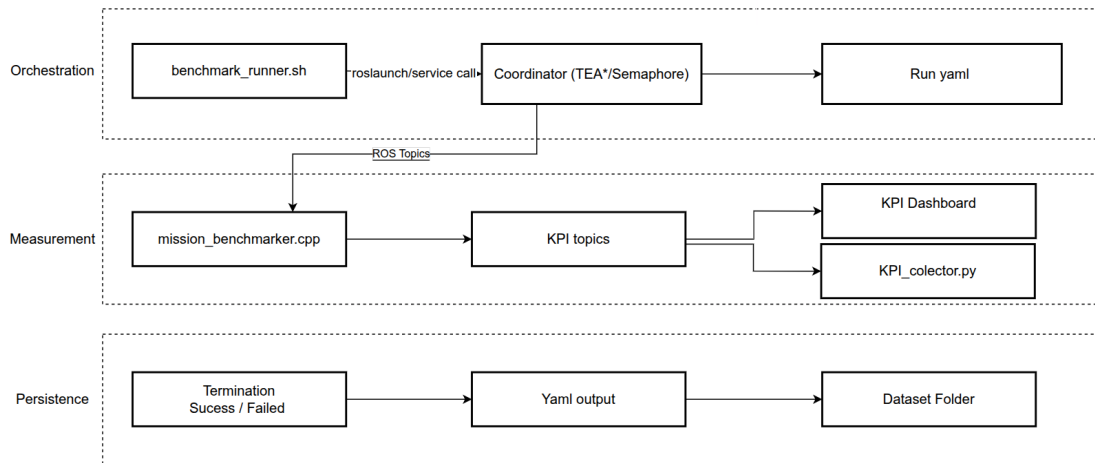


Figure 5.1: Architecture of the automated benchmarking pipeline.

feed a real-time KPI dashboard (see Appendix A.1), an rqt visualisation plugin used for live monitoring during a run.

To enable fully automated data collection, a dedicated Python node, the `kpi_collector`, was developed to replace the manual export step: it subscribes to the same KPIs topics, accumulates all metrics in memory throughout the run, and writes the YAML file automatically when a termination condition is detected.

The persistence layer is the export mechanism inside the `kpi_collector`, when the run terminates, either successfully or by a failure condition, the collector serialises the accumulated state into a structured YAML file and writes it to the dataset directory for the coordinator under test.

5.1.2 Run Lifecycle

Each run follows a fixed sequence of steps enforced by the orchestration script.

The script starts the simulator and the coordinator under test via `roslaunch`. After launch, the script waits for all robots to report their initial symbolic pose on `/rosout` before proceeding, ensuring that the fleet is fully initialised before any missions are issued.

Once the fleet is ready, the `kpi_collector` is started and the mission file for the current experimental condition is dispatched to the coordinator via a ROS service call. From this point, the `mission_benchmark` captures all coordination events in real time and publishes.

A run terminates under one of three conditions. Normal termination occurs when the `mission_benchmark` detects that all robots in the batch have completed their missions and publishes on `/kpi/total_execution_time`; the collector receives this signal, writes the YAML file with status `SUCCESS`, and exits. Automatic abort occurs when the accumulated failed planning count reaches the threshold of 120 invocations, at which point the collector writes the file with status `FAILED` (120+ failed plannings) and signals the orchestration script. Timeout abort occurs when

the maximum run duration (1200 seconds) elapses without a termination signal; the orchestration script sends an interrupt to the collector, which writes the file with status `FAILED (Physical Deadlock / Timeout)`.

After every run, regardless of outcome, the script kills all ROS processes and pauses before launching the next configuration, preventing resource leaks from affecting subsequent runs.

5.1.3 Output Structure

Each run produces one YAML file, written by the `kpi_collector` at termination. The file is organised in three blocks. The `benchmark_meta` block records the scenario identifier, fleet size, and run outcome. The `global_kpis` block records fleet-level aggregates: total missions, total replannings, average and total execution times, failed planning count, average planning time, and the last recorded scheduling priority list. The `robots` block records per-robot summaries and, for each mission, the execution time, final planned cost, replanning count, and the full `plan_history`. For each mission the `plan_history` holds one entry per planning attempt, the initial route and one per replanning event, each recording that route's planned cost. A complete example of such a file, illustrating the three-block layout, is provided in Appendix A.3.

Files are written to a directory structure organised by coordinator, scenario, and workload. This layout allows the benchmarking tool to load all runs for a given coordinator by selecting the corresponding directory.

5.2 The Node Analyzer

The `mission_benchmark` is a standalone ROS node (C++) that runs in parallel with the coordination system, subscribing to existing topics without interfering with the coordinator under test. It is responsible for capturing coordination events in real time and deriving the KPIs that feed the analysis pipeline.

5.2.1 Dynamic Robot Discovery

The node requires no prior configuration of fleet size. It subscribes to the `global/execution_route_states` topic and, upon each message, extracts the `request_name_space` field to identify the publishing robot. When a previously unseen robot is detected, three per-robot subscribers are created dynamically at runtime: `/target_route` to detect missions and replannings, `/navigation_handler2/execution_route_states` to initiate execution timers, and `/navigation_handler2/Go2vertex/goal` to record the first navigation timestamp for batch timing.

Taken together, these subscriptions define the interface contract that any coordinator must satisfy to be evaluated by the pipeline. A system under test must publish route assignments and replannings on `/target_route`, per-robot execution state on `/execution_route_states`, mission completion on the supervisor's `/RobotCompletion` topic, and expose the facility graph on `/server/graph`; failed planning invocations are read from the planner's log channel. No coordinator-specific code

is required; any new system meeting these interface requirements is evaluated by the same node without modification, making the methodology system-agnostic.

5.2.2 KPI Computation

Planned cost is computed when a robot receives a new route, by summing the temporal costs of the edges in the route sequence using a map populated from the facility graph published on `/server/graph`. The same calculation is repeated at each replanning event, and the result is stored per attempt in the `plan_history` structure.

Replanning detection uses the `request_id` field of incoming route messages. A new `request_id` registers a new mission; a repeated one increments the replanning counter for that mission and records the updated planned cost.

Execution time is measured between the stamp of the first `execution_route_state` for a mission and the `completion_time` from the path supervisor's `/RobotCompletion` message. Total batch execution time is the difference between the first `Go2vertex/goal` timestamp across any robot and the last completion time.

Failed plannings are detected by monitoring `/rosout` and filtering log messages from `/server/-path_planner_central` for a specific string that identifies each failed planner invocation. This avoids any modification to the coordinator's source code.

5.3 Semaphore Baseline

The semaphore baseline was developed in the scope of this dissertation specifically to serve as that reference. It implements the simplest form of traffic management capable of operating a fleet on a shared graph without any global plan or temporal look-ahead. It also serves to validate the benchmarking pipeline itself.

5.3.1 Operating Principle

When a mission is requested, the semaphore coordinator calls the same spatial A* path planner used by TEA* to obtain a sequence of edges from the robot's current position to the destination vertex. This route is then executed progressively. Before advancing to each edge, the coordinator attempts to acquire a set of locks: one on the edge itself, one on its destination vertex, and an additional lock on the structural group (such as a chokepoint or corridor) to which the edge belongs. Each lock is implemented as a semaphore object with a binary state (free or occupied) and a FIFO wait queue.

If all required locks are available, the robot receives the edge command and proceeds. If any lock is held by another robot, the requesting robot is placed in the wait queue for that resource and blocks. A lock is only released when the robot physically clears the corresponding space. Once clear, it releases the locks it held, and the next robot in each queue is granted access and notified to advance.

This mechanism is entirely reactive: each locking decision is made at the moment a robot reaches the boundary of the next edge, based only on the current occupancy state of the required locks. No future positions are considered, and no coordination occurs between robots beyond the implicit serialisation imposed by the queues.

5.3.2 Absence of Temporal Look-ahead and Deadlock Susceptibility

The fundamental difference between the semaphore coordinator and TEA* is the absence of temporal reasoning. TEA* plans routes in a time-extended graph where the future positions of all robots are known at planning time, allowing it to schedule routes that avoid conflicts before they occur [10]. The semaphore coordinator has no such model: it cannot predict whether a robot will still be occupying a resource when another robot arrives, nor can it reroute a robot to avoid a forthcoming conflict. It can only observe the present state of each lock and react accordingly.

This reactive nature makes the semaphore coordinator susceptible to circular wait conditions. If robot A holds the lock on edge e_1 and is waiting for edge e_2 , while robot B holds the lock on e_2 and is waiting for e_1 , neither robot can advance and the system enters a deadlock. Since each locking decision is made independently with no global state, this condition cannot be anticipated or resolved by the coordinator itself. At sufficiently high fleet densities, such circular dependencies become increasingly likely.

5.3.3 Integration with the Benchmarking Pipeline

The semaphore coordinator runs under the same ROS environment as TEA* and is evaluated through the identical benchmarking pipeline described in Section 5.1. The `mission_benchmark` subscribes to the same standardised topics in both cases: mission completion and route assignments are published on the same topic structure. No modifications to the measurement infrastructure are required to switch between coordinators, which ensures that all KPIs are computed by the same logic and are directly comparable.

5.4 Benchmarking Analysis Tool

Each test cycle produces dozens of YAML files spanning multiple coordinators, scenarios, fleet sizes, and workloads. The benchmarking analysis tool was developed to automate this process. It loads two datasets of YAML files, aggregates the metrics, and presents a set of comparative visualisations that translate raw KPI data.

5.4.1 Architecture

The tool is implemented as an rqt plugin in C++ and loaded directly inside the ROS rqt framework. The graphical interface is built entirely with Qt5 Widgets and Qt Charts without any external browser or web rendering engine. All visualisation is native Qt, which eliminates network dependencies and ensures stable rendering within the rqt environment.

The YAML files are parsed by a custom line-by-line reader using QTextStream without any external YAML library. Fields absent from a file receive silent default values, zero for numeric fields and empty string for text fields, and do not cause parsing errors. This makes the tool tolerant to partially complete runs.

5.4.2 Interface Structure

The interface is organised around three screens managed by a central `QStackedWidget`.

The Home screen (Figure 5.2) presents two entry points: Quick Compare and Detailed Analysis. Each is represented by a card that navigates to the corresponding mode.

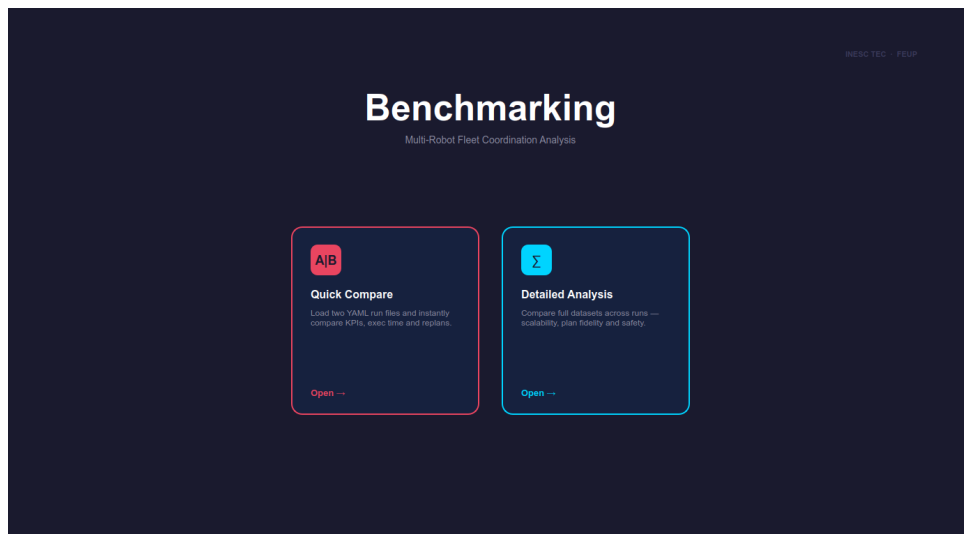


Figure 5.2: Home screen of the benchmarking analysis tool, presenting the two analysis modes.

The Quick Compare allows loading two individual YAML files, one per system, and produces an immediate side-by-side summary without requiring organised dataset folders. It is intended for rapid inspection of a single experimental condition.

The Detailed Analysis operates in two steps. The Loader screen (Figure A.2) provides two folder pickers, one per dataset. The analysis is triggered once both folders are selected and all YAML files within them have been parsed. The tool then navigates to the Dashboard screen, which is the primary analysis interface composed of a scenario filter and five tabs described in Section 5.4.3.

5.4.3 Analytical Views

The following subsections describe the content and layout of each mode in detail.

5.4.3.1 Quick Compare

Given two individual YAML files, Quick Compare presents a single view that consolidates all per-run KPIs. At the top (Figure 5.3), a global performance summary table lists total missions, total

replannings, average execution time, failed plannings, average planning time, and total execution time for each run, together with the percentage difference. Below the table, per-robot bar charts compare replannings and average execution time side by side.



Figure 5.3: Quick Compare: global KPI summary table and per-robot bar charts.

Scrolling further, a mission execution breakdown (Figure 5.4) allows the user to select any robot and inspect individual missions. Each mission card shows the execution time of both runs, the percentage difference, the replanning count, and the planned costs.

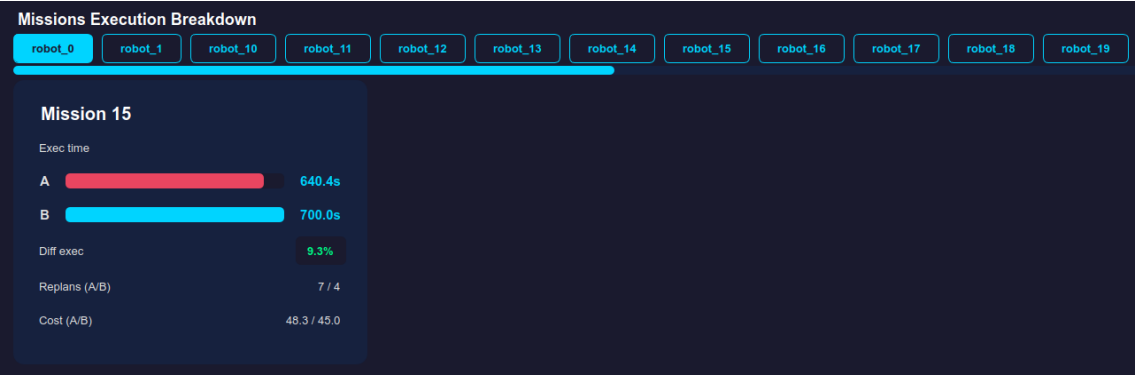


Figure 5.4: Quick Compare: per-mission execution breakdown with execution time and cost comparison.

A per-robot comparison table (Figure 5.5) lists every robot with its execution time and replanning count under both runs, computing efficiency and replanning differences as signed percentages.

Robot	Run A Exec	Run B Exec	Run A Replans	Run B Replans	Eff Diff	Replans Diff
robot_0	640.40	730.40	7	4	+14.1%	-42.9%
robot_1	633.50	610.50	7	2	-3.6%	-71.4%
robot_10	628.30	590.30	7	5	-6.0%	-28.6%
robot_11	651.90	640.90	7	6	-1.7%	-14.3%
robot_12	629.70	615.70	7	3	-2.2%	-57.1%
robot_13	622.30	610.30	7	9	-1.9%	+28.6%
robot_14	634.50	620.50	7	4	-2.2%	-42.9%
robot_15	619.20	600.20	7	3	-3.1%	-57.1%
robot_16	610.70	590.70	7	5	-3.3%	-28.6%
robot_17	638.60	620.60	7	4	-2.6%	-42.9%
robot_18	615.10	600.10	7	2	-2.4%	-71.4%
robot_19	633.30	620.30	7	6	-2.1%	-14.3%
robot_2	630.60	610.60	7	5	-3.2%	-28.6%
robot_20	621.10	610.10	7	4	-1.8%	-42.9%

Figure 5.5: Quick Compare: per-robot comparison table with efficiency and replanning differences.

This mode is useful for verifying a specific run or investigating an anomaly without loading a full dataset.

5.4.3.2 Detailed Analysis

The dashboard is organised into five tabs and includes a global scenario filter that allows the user to isolate data from specific simulation layouts. Each tab aggregates the relevant KPIs from all loaded runs.

The Overview tab (Figure 5.6) presents global aggregate metrics: average total execution time, average replanning events, average failed plannings, and average planning time. It provides an immediate summary of the relative performance of the two systems across all loaded conditions.



Figure 5.6: Detailed Analysis: Overview tab presenting global aggregate KPIs.

The Scalability tab (Figure 5.7) groups runs by fleet size N and plots average total execution time and total replanning events as line charts against N for both systems. It includes a dedicated workload filter to evaluate how the fleet scales under different task generation rates. Runs classified as `FAILED` are excluded from the execution time average but included in the replanning count, and the line breaks at fleet sizes where all runs failed, making system collapse visually explicit.



Figure 5.7: Detailed Analysis: Scalability tab visualising execution time and replanning trends against fleet size.

The Plan Fidelity tab (Figure 5.8) renders a scatter plot of planned cost versus actual execution time for every individual mission across all runs. Points above the diagonal indicate execution slower than planned. A per-robot fidelity ratio table is also computed.



Figure 5.8: Detailed Analysis: Plan Fidelity tab showing the correlation between planned cost and execution time.

The Replanning tab (Figure 5.9) presents replanning events as a function of fleet size and as a per-robot breakdown, allowing the distribution of replanning load across the fleet to be inspected.



Figure 5.9: Detailed Analysis: Replanning tab showing the distribution of replanning load across the fleet.

The Safety tab (Figure 5.10) presents the total number of failed plannings and plots the operational success rate as a function of fleet size for both systems, alongside the average failed planning count per N .



Figure 5.10: Detailed Analysis: Safety tab comparing the operational success rate of both systems.

Chapter 6

Experimental Results

This chapter presents the data collected by the benchmarking pipeline, detailing the experimental setup and reporting the results for scalability, plan fidelity, replanning frequency, and operational safety.

6.1 Experimental Setup

All runs were executed on the same machine under ROS, using the automated pipeline described in Section 5.1. Both coordinators were evaluated under identical conditions: same simulator configuration, same mission seeds per experimental condition, and same benchmarking infrastructure. This ensures that all observed differences in KPI values are attributable to coordinator behaviour and not to extraneous variation in the experimental setup.

6.1.1 Experimental Conditions

Two scenarios were evaluated, each with a different set of fleet sizes and two workload configurations.

The two workload sets differ in the number of tasks assigned per mission. In `m1`, each robot is assigned a single task: navigating to a target vertex. In `m2`, each robot is assigned two sequential tasks: navigating to a target vertex and then returning to the starting vertex. The second workload therefore doubles the travel demand per mission and increases traffic density in the bottleneck regions of the graph.

For `sim1_portic`, fleet size was varied across six levels: $N \in \{2, 4, 8, 12, 16, 20\}$, combined with both workload sets, yielding 12 experimental conditions per coordinator.

For `sim2_hosp`, fleet size was varied across seven levels: $N \in \{2, 3, 4, 5, 6, 7, 8\}$, combined with both workload sets, yielding 14 experimental conditions per coordinator.

One run was executed per experimental condition per coordinator, producing a total of 52 runs: 26 for TEA* and 26 for the semaphore coordinator.

6.2 Scalability Results

This section presents the total execution time as a function of fleet size N for both coordinators across the two test scenarios. These results provide the empirical basis for evaluating hypotheses H1 (scalability) and H2 (bottleneck amplification).

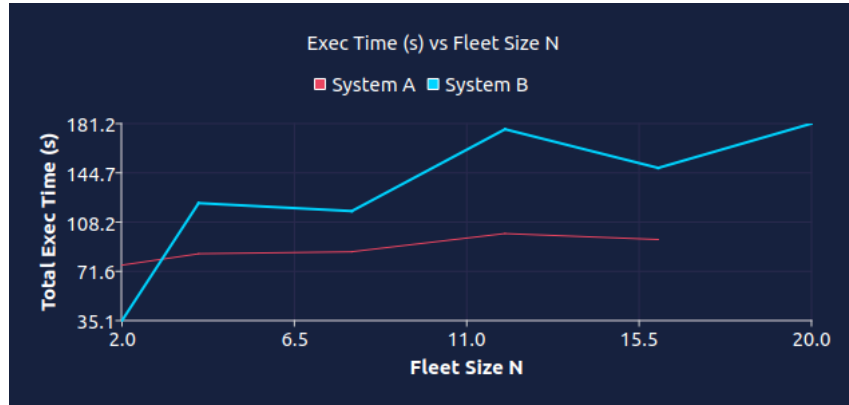
6.2.1 Industrial Facility Scenario

Table 6.1 reports the total execution time for each experimental condition in the industrial facility scenario, which features a partially open layout with moderate bottlenecks and sufficient alternative routing options for both coordinators to operate at low fleet sizes. The semaphore coordinator completes all runs under workload `m1` up to $N = 16$, failing at $N = 20$. Under workload `m2`, it fails at $N = 12$, $N = 16$, and $N = 20$. TEA* completes all runs under `m1` and all runs under `m2` except $N = 16$, which is the single `FAILED` run recorded for TEA* in this scenario.

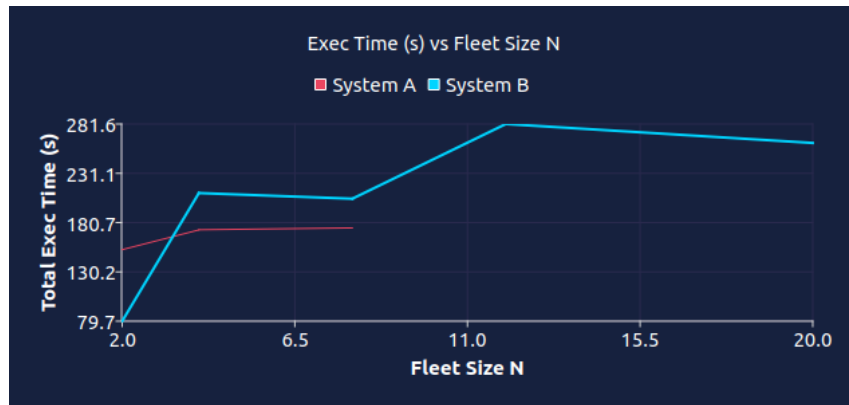
Table 6.1: Total execution time (seconds) in `sim1_portic`.

N	TEA*		Semaphore	
	<code>m1</code>	<code>m2</code>	<code>m1</code>	<code>m2</code>
2	35.1	79.7	76.3	153.1
4	122.2	210.9	84.7	173.3
8	116.3	205.1	86.1	175.2
12	177.0	281.6	99.6	FAILED
16	148.3	FAILED	95.2	FAILED
20	181.2	262.2	FAILED	FAILED

TEA* remains operational across all fleet sizes under `m1`, with execution times growing from 35.1 s at $N = 2$ to 181.2 s at $N = 20$. Under `m2`, execution times are consistently higher and the single `FAILED` run occurs at $N = 16$. Among the fleet sizes it completes, the semaphore coordinator records execution times below those of TEA* at $N = 4$ and $N = 8$, though not at $N = 2$, where TEA* is faster; it begins to fail under `m2` at $N = 12$ and fails under both workloads at $N = 20$. Figure 6.1 shows the corresponding scalability curves.



(a) Workload m1.



(b) Workload m2.

Figure 6.1: Total execution time (s) vs fleet size N in the industrial facility scenario. TEA* (System B), semaphore (System A). Missing points indicate FAILED runs.

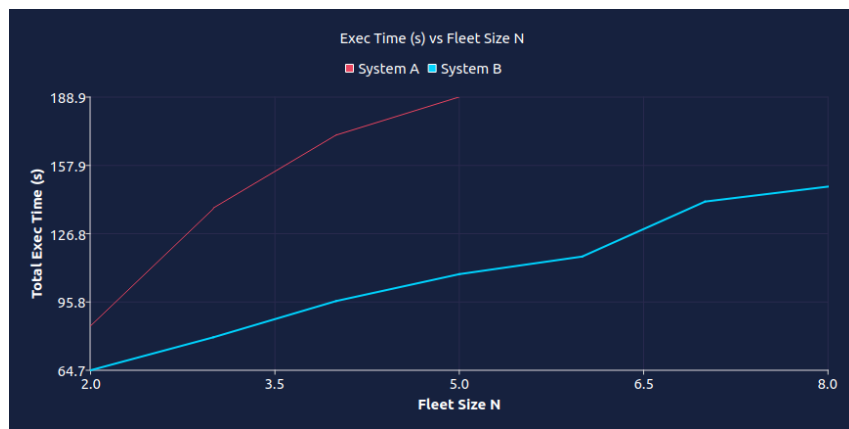
6.2.2 Hospital Facility Scenario

Table 6.2 reports the total execution time for the hospital facility scenario, which features narrow corridors, a high density of single-access nodes, and fewer alternative routes, conditions that impose substantially greater coordination pressure than the industrial scenario. The semaphore coordinator successfully completes runs up to $N = 5$ under the m1 workload, but struggles under the m2 workload, where it only succeeds at $N = 2$ and fails for all larger fleet sizes. In contrast, TEA* completes all conditions under m1 across the full range $N \in \{2, \dots, 8\}$, but under m2 it fails at $N = 6$, $N = 7$, and $N = 8$.

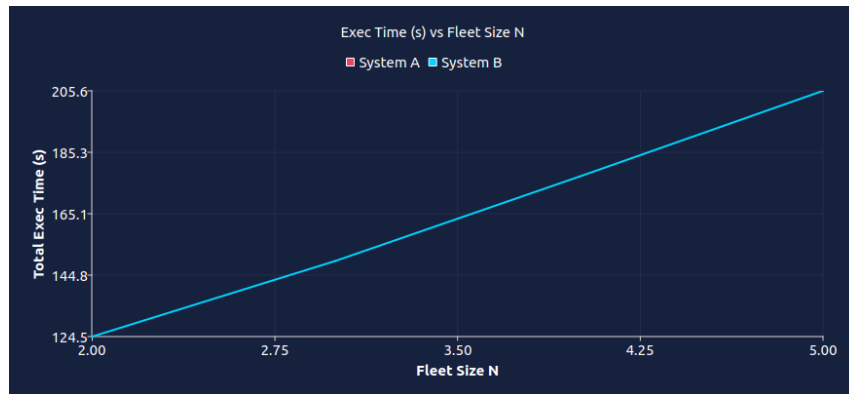
Among the successful runs, TEA* demonstrates steady execution times under m1, scaling from 64.7 s at $N = 2$ to 148.2 s at $N = 8$. In contrast, the semaphore approach exhibits a steeper degradation under the same workload, rising from 85.2 s to 188.9 s ($N = 5$). For the more demanding m2 workload, TEA* execution times range from 124.5 s to 205.6 s ($N = 5$). Meanwhile, the semaphore is overwhelmed almost immediately under m2, taking 233.6 s to complete the $N = 2$ case. Figure 6.2 shows the execution time versus fleet size for each workload.

Table 6.2: Total execution time (seconds) in `sim2_hosp.`

N	TEA*		Semaphore	
	m1	m2	m1	m2
2	64.7	124.5	85.2	233.6
3	79.7	149.5	138.4	FAILED
4	96.2	177.3	171.6	FAILED
5	108.4	205.6	188.9	FAILED
6	116.4	FAILED	FAILED	FAILED
7	141.4	FAILED	FAILED	FAILED
8	148.2	FAILED	FAILED	FAILED



(a) Workload m1.



(b) Workload m2.

Figure 6.2: Total execution time (s) vs fleet size N in the hospital facility scenario. TEA* (System B), semaphore (System A). Missing points indicate FAILED runs.

6.3 Plan Fidelity Results

This section presents the relationship between planned cost and actual execution time for each mission recorded across all completed runs. These results provide the empirical basis for evaluating

hypothesis H4.

6.3.1 Industrial Facility Scenario

Figure 6.3 shows the scatter plot of planned cost versus real execution time for all individual missions in the industrial facility scenario. The dashed diagonal represents the ideal ratio of unity, where execution time equals planned cost exactly. Points above the diagonal indicate missions where execution took longer than planned.

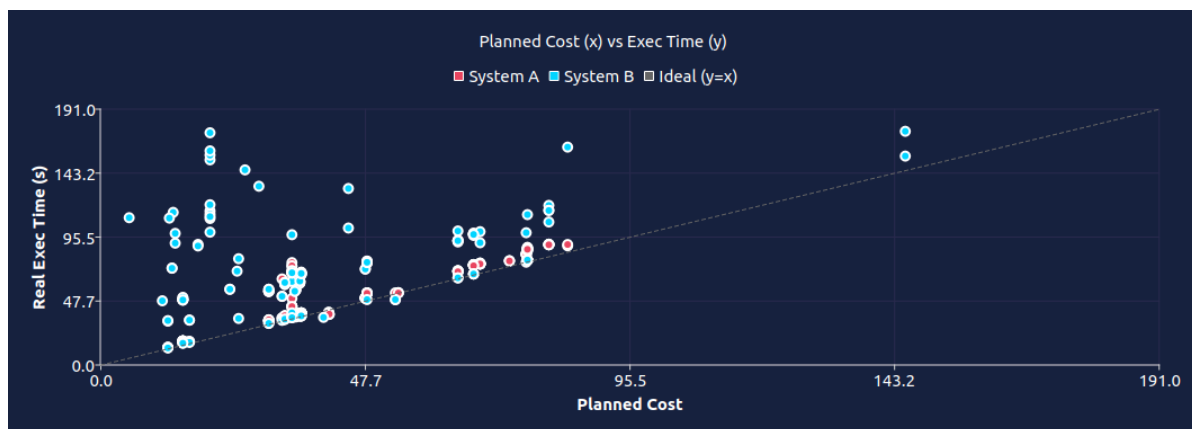


Figure 6.3: Planned cost vs real execution time for all missions in the industrial facility scenario. Semaphore (System A), TEA* (System B). Dashed line: ideal ratio $y = x$.

The semaphore coordinator's points cluster near or below the diagonal across the full range of planned costs, with fidelity ratios ranging from $\times 0.99$ to $\times 1.47$, indicating that execution times are generally consistent with the planned route cost. TEA* presents a different distribution, its points are systematically positioned above the diagonal, with ratios ranging from $\times 1.26$ to $\times 5.03$ and substantially wider scatter, indicating higher variability in the relationship between planned cost and actual execution time. The complete per-robot fidelity ratio tables for both scenarios are provided in Appendix A.4.

6.3.2 Hospital Facility Scenario

Figure 6.4 shows the scatter plot for the hospital facility scenario. Reflecting its successful runs up to $N = 5$, the semaphore coordinator's data includes five robots, with fidelity ratios ranging from $\times 1.81$ to $\times 3.31$. TEA* presents substantially higher ratios in this scenario, ranging from $\times 1.89$ to $\times 19.04$. Several robots present planned cost values near zero while execution times remain high, producing points clustered in the upper-left region of the scatter plot, far above the diagonal. This pattern is more pronounced than in the industrial facility scenario.

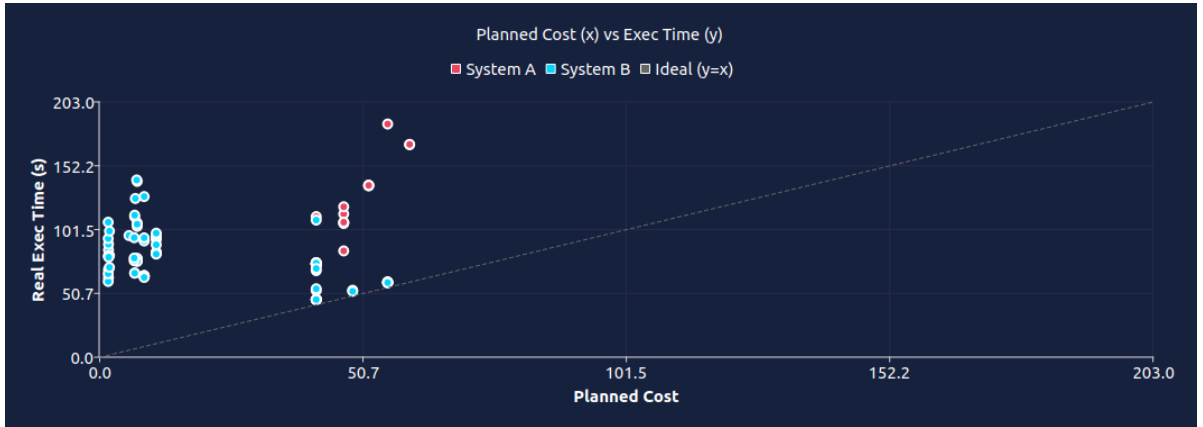


Figure 6.4: Planned cost vs real execution time for all missions in the hospital facility scenario. Semaphore (System A), TEA* (System B). Dashed line: ideal ratio $y = x$.

6.4 Replanning Analysis

This section presents the replanning events recorded across all runs for both coordinators. These results provide the empirical basis for evaluating hypothesis H3.

The semaphore coordinator produces zero replanning events by design across all experimental conditions in both scenarios, as it has no supervisor layer and resolves conflicts exclusively through reactive edge locking. All replanning activity reported in this section therefore corresponds exclusively to TEA*.

6.4.1 Industrial Facility Scenario

Figure 6.5 shows the average total replanning events as a function of fleet size N for TEA* in the industrial facility scenario. The count grows with N , from near zero at $N = 2$ to approximately 421 at $N = 20$, with a notable acceleration between $N = 16$ and $N = 20$.

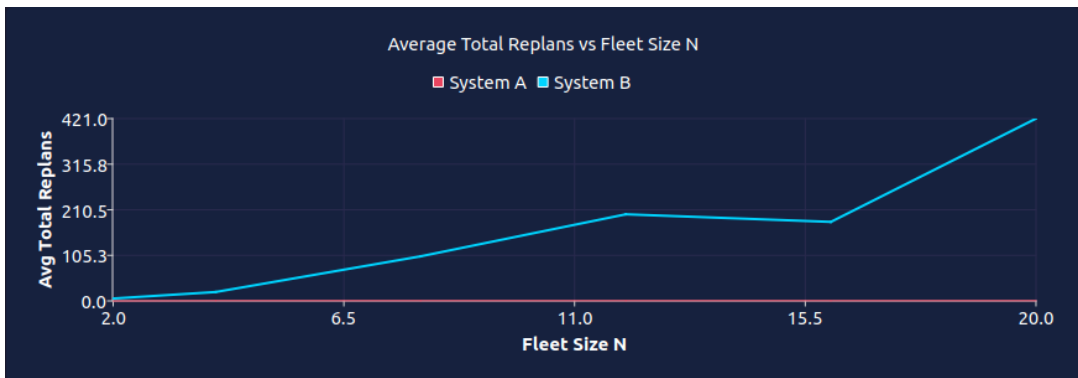


Figure 6.5: Average total replanning events vs fleet size N . TEA* (System B) and semaphore (System A).

Figure 6.6 shows the average replanning events per robot across all runs and missions. The distribution is uneven across the fleet: robot_9 and robot_18 present the highest averages (18.00 and 19.67 respectively), while robot_11 presents the lowest (3.67). This indicates that replanning load is not uniformly distributed across robots. The complete per-robot detail is provided in Appendix A.5.

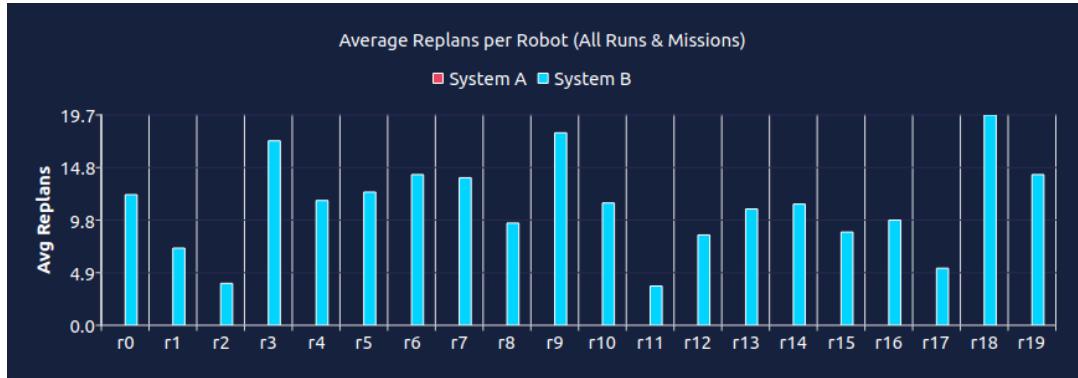


Figure 6.6: Average replanning events per robot across all runs and missions. TEA* (System B) and semaphore (System A).

6.4.2 Hospital Facility Scenario

Figure 6.7 shows the replanning trend in the hospital facility scenario. The count rises overall from $N = 2$ to $N = 8$, with a local peak at $N = 5$ followed by a temporary decrease before resuming growth. The overall range of replanning events in this scenario is lower than in the industrial facility, consistent with the smaller fleet sizes tested.

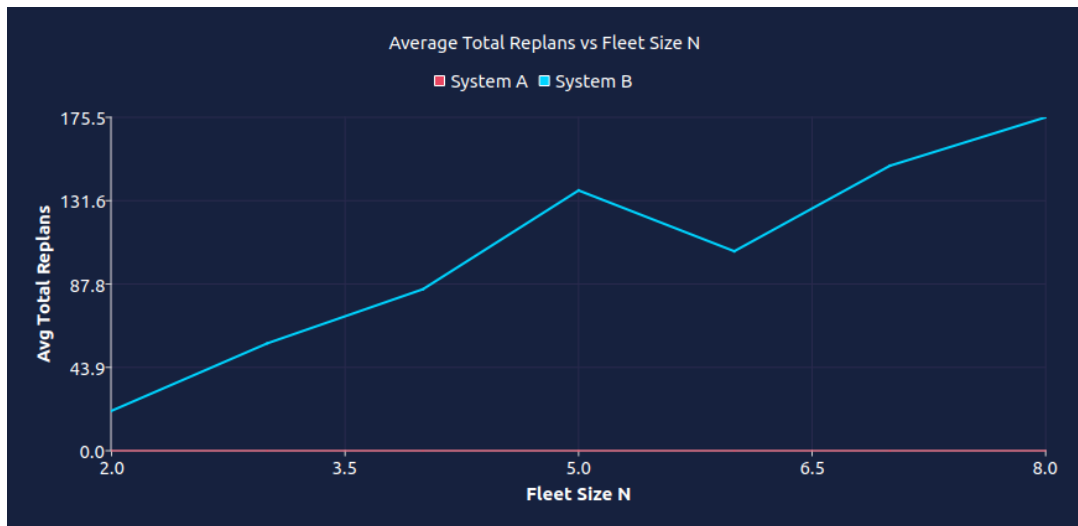


Figure 6.7: Average total replanning events vs fleet size N . TEA* (System B) and semaphore (System A).

Figure 6.8 shows the per-robot distribution in the hospital scenario. Compared to the industrial facility, the distribution across robots is more uniform, with values ranging from approximately 12 to 19 replans per robot. The semaphore coordinator records zero replanning events for all robots.

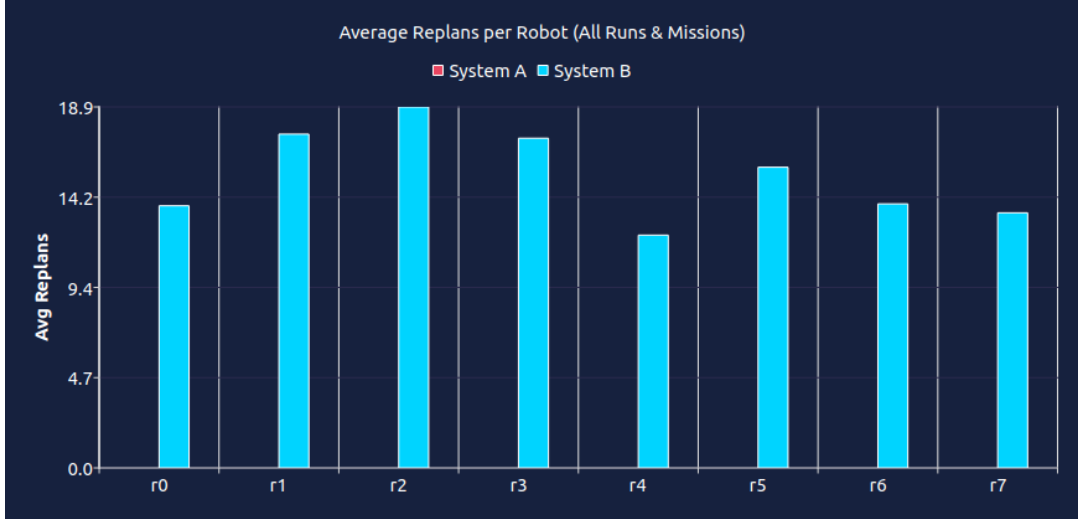


Figure 6.8: Average replanning events per robot across all runs and missions. TEA* (System B) and semaphore (System A).

6.5 Safety Results

This section presents the failed planning events and operational success rates recorded across all runs for both coordinators. These results provide the empirical basis for evaluating hypothesis H5.

The two coordinators exhibit fundamentally different failure modes. TEA* resolves conflicts during planning, in a time-extended search space. When coordination pressure exceeds the planner’s capacity, no valid time window exists for all robots to proceed without conflict, the search exhausts its options and a planning failure is recorded. The semaphore coordinator, by contrast, plans each robot’s route independently of all others using spatial A*, which always finds a path as long as the graph is connected. Planning therefore never fails for the semaphore. Instead, failures are deferred to execution, robots proceed until they are physically blocked by others waiting for locked edges, producing circular wait conditions that no robot can resolve. The run then times out as a physical deadlock.

6.5.1 Industrial Facility Scenario

As expected, the semaphore coordinator records zero failed planning events across all runs in this scenario. TEA* accumulates 391 total failed planning invocations across its dataset. Missions completed for the semaphore total 163 out of 179, versus 158 out of 171 for TEA*, a difference of 1.3%.

Figure 6.9 shows the average failed planning count as a function of fleet size N for both coordinators. TEA* shows a steep increase from $N = 2$, reaching approximately 48.5 failed plans per run at $N = 20$, while the semaphore remains at zero throughout.

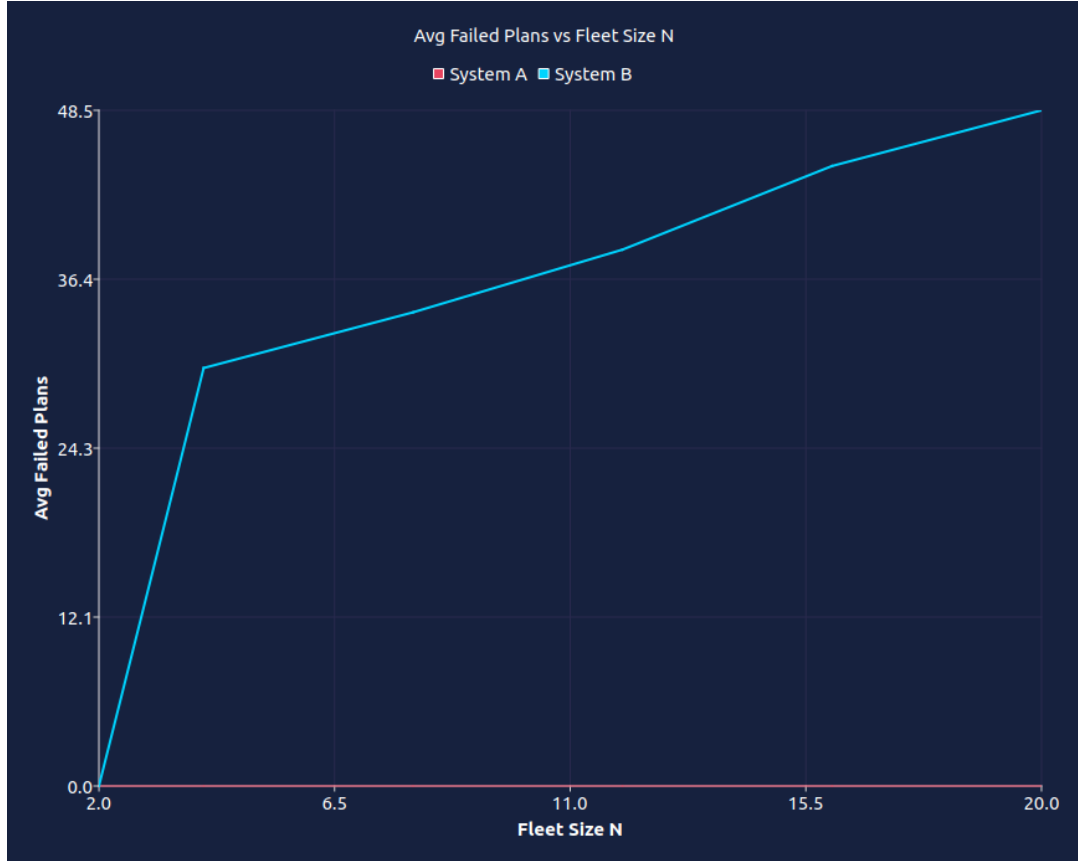


Figure 6.9: Average failed planning events vs fleet size N . TEA* (System B), semaphore (System A).

Figure 6.10 shows the operational success rate, the fraction of completed runs, as a function of N . The semaphore coordinator maintains a 100% success rate up to $N = 8$ under both workloads, after which it degrades: it reaches 50% at $N = 12$ and falls to 0% at $N = 20$. TEA* maintains 100% up to $N = 12$, drops to 50% at $N = 16$ due to the single FAILED run under workload m2, and returns to 100% at $N = 20$, where both workloads complete. The curves diverge significantly at high fleet sizes, with the semaphore collapsing earlier than TEA*.

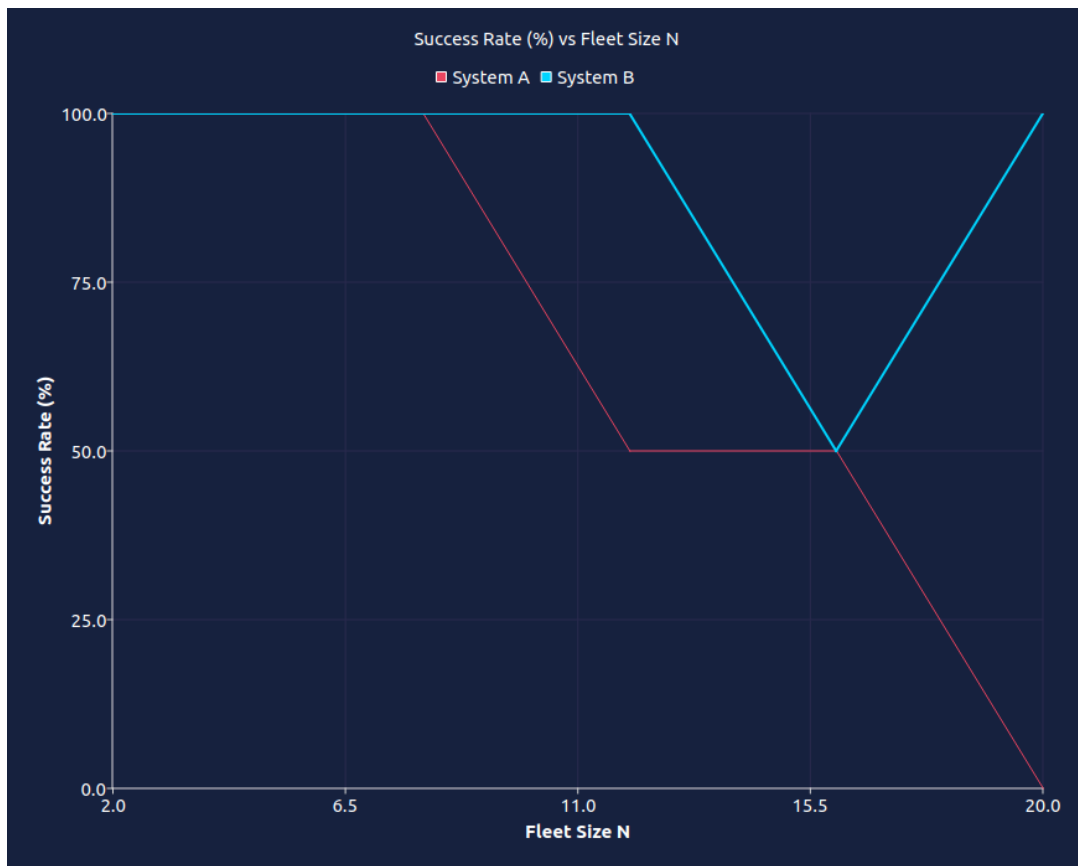


Figure 6.10: Operational success rate (%) vs fleet size N . Each point represents the fraction of completed runs at that fleet size. TEA* (System B), semaphore (System A).

6.5.2 Hospital Facility Scenario

As expected, the semaphore coordinator records zero failed planning events in this scenario. TEA* accumulates 21 total failed planning invocations. Missions completed for the semaphore total 27 out of 78, versus 73 out of 95 for TEA*, a difference of 42.2%, reflecting the semaphore's very limited operational range in this scenario.

Figure 6.11 shows the failed planning trend for this scenario. TEA* records near zero failed plannings at $N = 2$ and $N = 3$, reaching approximately 3.5 at $N = 8$. The values are substantially lower than in the industrial facility scenario, consistent with the smaller fleet sizes evaluated.

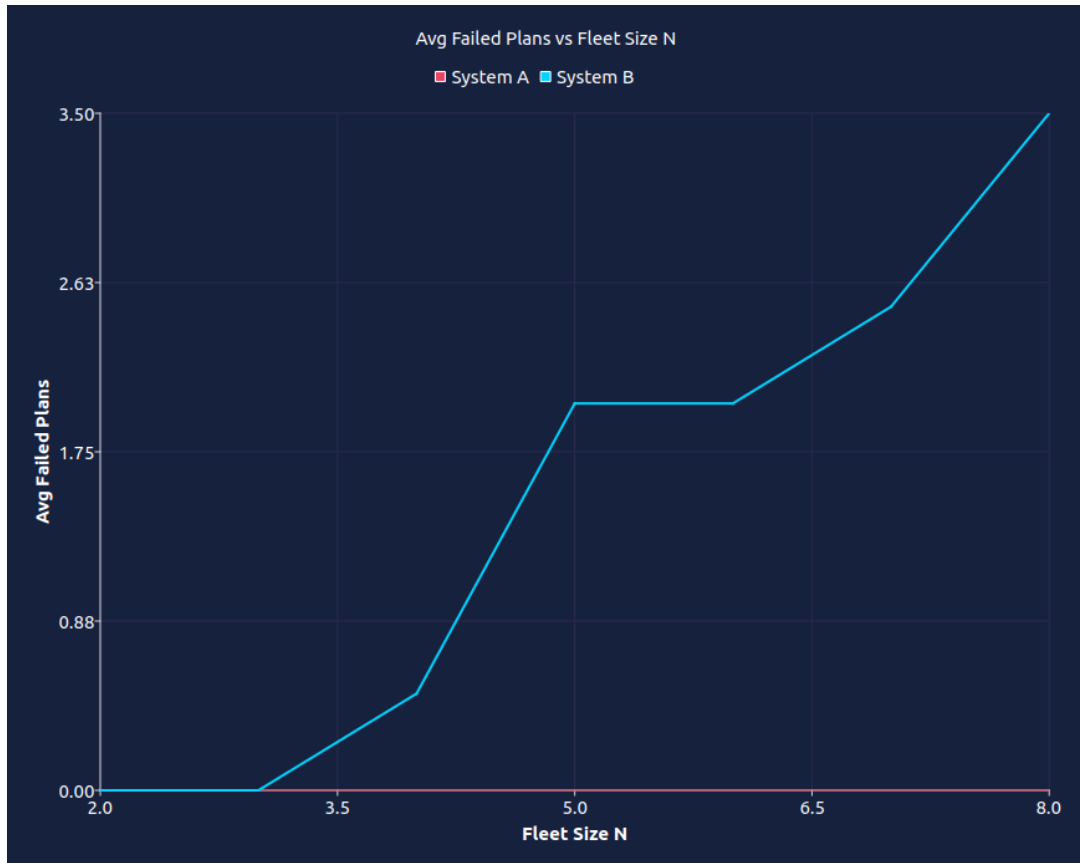


Figure 6.11: Average failed planning events vs fleet size N . TEA* (System B), semaphore (System A).

Figure 6.12 shows the success rate for this scenario. The semaphore coordinator achieves 100% success only at $N = 2$, drops to 50% between $N = 3$ and $N = 5$ where it completes only workload m_1 , and collapses to 0% from $N = 6$ onwards. TEA* maintains 100% success up to $N = 5$, after which it drops to 50% at $N = 6$ and stabilises at 50% through $N = 8$, due to failures under workload m_2 at those fleet sizes. The gap between the two success rate curves is substantially wider in this scenario than in the industrial facility, consistent with the greater structural coordination pressure imposed by the narrow-corridor layout.

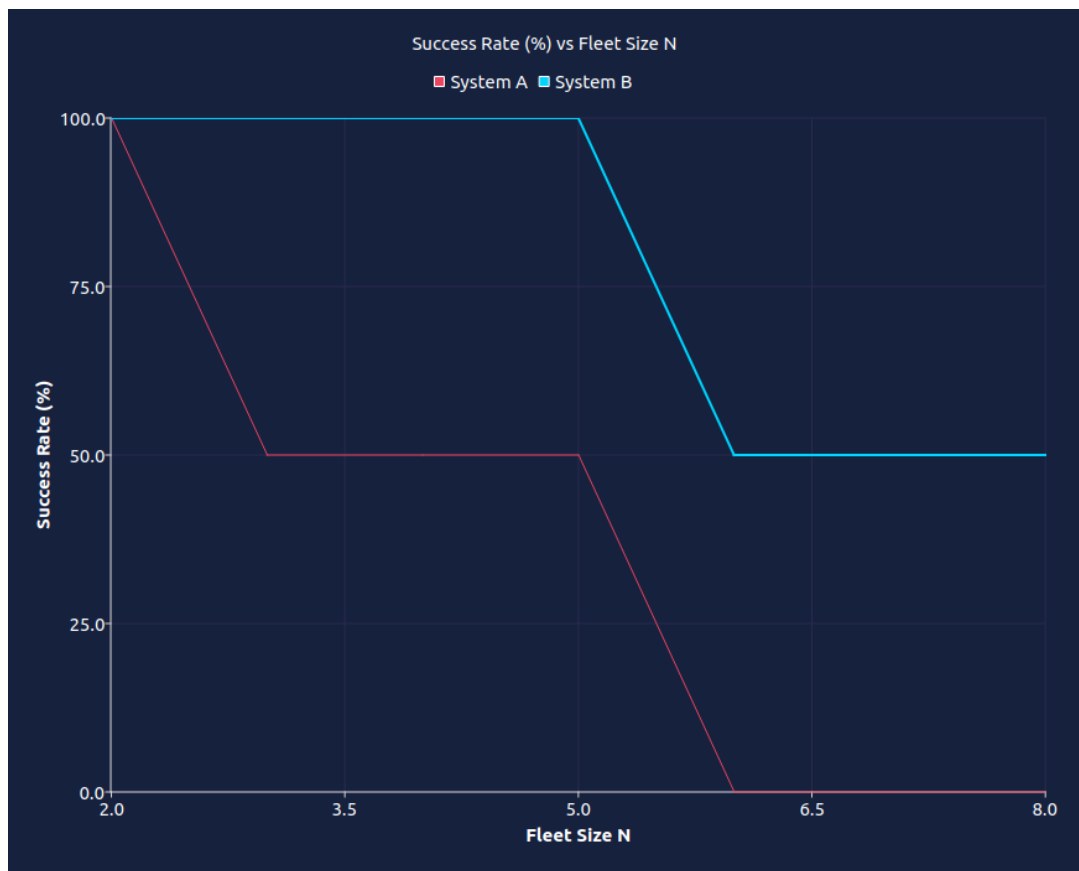


Figure 6.12: Operational success rate (%) vs fleet size N . Each point represents the fraction of completed runs at that fleet size. TEA* (System B), semaphore (System A).

Chapter 7

Discussion

This chapter interprets the experimental results produced by the benchmark, presented in Chapter 6, in light of the five scientific hypotheses formulated in Section 4.3.

7.1 The Benchmark Tool

The object proposed by this dissertation is the benchmark, the pipeline, the KPI schema, and the analysis tool that together turn a fleet-coordination run into a comparable, machine-readable record. The comparison between TEA* and the semaphore coordinator is what that instrument made possible. Each hypothesis tests one behaviour, and the corresponding result depends on a specific capability of the benchmark.

The H4 plan-fidelity result depends on the tool logging planned cost and execution time for the same mission across every replanning cycle. Without the per-mission `plan_history`, the gap between the route estimate cost and the execution time would not be visible. H2 holds only because the scenario is a controlled factor, so the same fleet sizes are run under different layouts and the wider gap can be attributed to the layout alone. H1 and H5 need the pipeline to tell a successful run from a failed one.

These conclusions could not be drawn from one run, one scenario, or one layer on its own. They come from measuring solution cost, planning time, replanning, and plan fidelity together, from the same run, across two scenarios and two workloads.

Beyond the results presented here, the benchmark has a continuing role in the development of the system. Because the pipeline is agnostic to the internal logic of the coordinator, any future change to the TEA* stack can be re-evaluated through the same instrument without adapting the measurement infrastructure. A modification that improves one KPI, such as reducing replanning frequency, can be checked immediately against the others, such as execution time and success rate, to verify that the gain is real and not merely a shift of the overhead from one layer to another. This is precisely the feedback loop that a benchmarking methodology is designed to provide.

7.2 Scalability: Time-Aware Planning Bounds Execution Growth

H1 is partially refuted. The prediction that TEA* achieves lower total execution times than the semaphore as N increases holds in the constrained hospital layout, where TEA* is consistently faster across all fleet sizes tested. In the open industrial layout, scaling the fleet inflates the total execution time of TEA* by a modest factor relative to the growth in N (Tables 6.1 and 6.2). The prediction that the semaphore is slower than TEA* does not universally hold. In the open industrial layout, its execution time stays low and almost flat before it collapses into deadlock without an intermediate regime of rising cost. However, in the hospital layout, it exhibits a steep performance degradation (rising sharply before failing at $N = 6$) and is consistently slower than TEA* (Section 6.2). The scalability advantage of TEA* originates both from contained, more stable execution-time growth under pressure, and from its ability to sustain much larger fleet sizes without collapsing.

This contained growth follows from time-extended planning. Reasoning in space-time, TEA* sequences robots through a shared resource in separate time windows using wait and give-way actions, instead of forcing exclusive occupancy [10], [17]. Rather than waiting in a queue, each new robot is fitted into gaps in the existing schedule, which keeps the added execution time small. This is paid for in waiting and supervisor-triggered replanning, which raises the absolute execution time but keeps its growth bounded.

The semaphore exposes a mechanism that H1 did not anticipate. It locks each edge, and its destination vertex only when a robot reaches it, with no model of future occupancy (Section 5.3). At low density, and especially in the open industrial layout, locks rarely contend, so robots barely wait and the coordinator is cheap, undercutting TEA* because it carries no planning overhead. As density increases, the missing look-ahead inevitably produces circular waits that a reactive rule cannot detect or break. In open layouts, this causes the run to abruptly stop rather than slow down. In constrained layouts, the system first suffers a rapid escalation in waiting time before collapsing. The predicted contention is real, but it ultimately resolves into deadlock rather than a smooth, progressively rising cost curve.

This behaviour matches the literature. Centralised, plan-based coordination is known to sustain throughput at fleet scale [1], [2], and TEA* obtains this without optimal conflict search, which scales poorly as conflicts multiply [7], [23], by trading optimality for incremental, time-aware resolution. The semaphore instead reproduces the deadlock mode of reactive mutual-exclusion schemes, which are incomplete in the non-well-formed, choke-point rich layouts [4], [16]. The two approaches differ fundamentally in whether the fleet runs at all, and the threshold is set by structural contention: the semaphore survives up to $N = 16$ in the open industrial layout but collapses at much smaller fleet sizes ($N \geq 6$) in the constrained hospital one. While execution time shows a clear difference under pressure, the most critical gap is their operational success, discussed under H2 and H5.

7.3 Bottleneck Amplification: Structural Contention Widens the Gap

H2 is confirmed. The separation between the two coordinators is markedly wider in the hospital scenario than in the industrial one, at matched fleet sizes and workloads. In the open industrial layout the semaphore coordinator remains operational up to $N = 16$ under the single-task workload before failing (Table 6.1), whereas in the hospital layout its operational range shrinks significantly, sustaining fleets only up to $N = 5$ under the same workload before collapsing (Table 6.2). The same divergence appears in the operational success curves, which track each other across most of the industrial range but separate sharply in the hospital scenario (Figures 6.10 and 6.12), and in the mission-completion difference, which is minor in the industrial facility (Section 6.5.1) but large in the hospital one (Section 6.5.2).

As established under H1, the most important difference between the systems is the maximum fleet size they can handle before completely failing. This confirms the prediction of H2: the constrained hospital layout severely amplifies coordination pressure, causing the semaphore's limit to drop from $N = 16$ in the open industrial layout down to a maximum of $N = 5$.

The mechanism is structural. The hospital layout combines narrow corridors with single-access nodes and offers few alternative routes, so a far higher fraction of robots is forced to contend for the same vertices at the same time. In a single-access corridor, the semaphore allows two robots approaching from opposite ends to each acquire a lock the other needs, producing a circular wait that a reactive rule cannot detect or unwind. TEA* resolves the same physical bottleneck in space–time, sequencing robots through the contested node in disjoint time windows by means of wait and give-way actions, so the corridor is shared temporally rather than competed for spatially [10], [17]. In the open layout the missing look-ahead stays hidden because spatial detours absorb most contention; in the hospital layout, where such detours do not exist, temporal resolution becomes the only viable mechanism.

This is the prediction made in Section 3.3. The literature identifies choke-point-dominated layouts, narrow corridors, merges, and doors, as exactly the regime in which purely spatial detours may not exist and time-slotting becomes the only resolution mechanism, which is what makes space–time planners particularly relevant for intralogistics [10], [16]. The hospital results confirm this empirically: the value of temporal reasoning is not a fixed offset but scales with structural contention, consistent with the expectation that realistic, cluttered environments amplify the difficulty of multi-robot coordination [11].

The advantage of a time-aware coordinator scales with environment severity. In open spaces the choice between the two approaches is close to inconsequential for whether the fleet completes its missions; in constrained corridors it determines feasibility outright. For benchmarking practice this means that evaluating a coordinator in a single, permissive scenario can conceal the very capability that distinguishes it, and that bottleneck severity must be treated as a controlled experimental factor rather than a fixed property of the test environment.

7.4 Replanning Overhead: Bounded Recovery, Not Fragility

H3 is confirmed. The replanning count of TEA* grows with fleet size in both scenarios, monotonically in the open industrial layout and with a steadier upward trend in the constrained hospital one (Figures 6.5 and 6.7). The semaphore coordinator records none of these events, by design (Section 6.4), and is read here as a structural reference rather than a comparator, since the two systems are incommensurable on this KPI.

TEA* uses replanning as its primary conflict-resolution mechanism during execution. The supervisor tracks the *Delta_steps* deviation between agent pairs relative to their planned trajectories and re-invokes the planner once a pair drifts past its threshold [10]. More robots mean more interacting pairs and denser traffic at the same bottleneck vertices, so the number of interventions rises with N . The increase stays bounded because each robot is interleaved into the existing space–time schedule and the supervisor fires selectively, on Danger Close pairs near shared resources, not on every possible interaction [10].

The conflicts that TEA* dissolves through replanning are precisely those the semaphore cannot resolve: with no look-ahead it accumulates them until a circular wait forms and the run deadlocks, the failure mode established under H1 and H2. Its replans are a signature of resilience, the visible cost of keeping the fleet moving where a reactive rule would stall it. This matches the known incompleteness of reactive mutual-exclusion schemes in choke-point-rich layouts [16] and the report that the *Delta_steps* supervisor completed all tested runs without deadlock [10].

Much of the coordination load is absorbed before it reaches the supervisor. Reasoning in space–time, TEA* anticipates conflicts at planning time and inserts wait and give-way actions, so replanning is reserved for the residual desynchronisation that the temporal cost model did not foresee, the delays that accumulate during real execution [17], [19]. The literature offers an alternative that avoids runtime replanning altogether, order-preserving execution policies such as MCP, but these require at least a 1-robust (MAPF-DP) base plan, in TEA* robustness to delays is handled at execution time by the supervision layer rather than embedded in the plan [19]. TEA* therefore pays for adaptivity in recovery events rather than in pre-planned slack, and the acceleration of the replanning curve at the largest industrial fleets (Figure 6.5) marks the onset of the replanning-cascade regime, where recovery cost approaches the validity horizon of the plans being computed [10].

Replanning count is an overhead KPI, not a failure indicator. A coordinator that replans often yet completes its missions is more robust than one that never replans and deadlocks; the relevant question is not how the count compares to the semaphore’s zero, but whether it scales sustainably with N . Both scenarios confirm this, though at the top of the industrial range the recovery load grows to the point where it outpaces the planner, which is what produced the isolated failed run in H5.

7.5 Plan Fidelity: The Logged Planned Cost Underestimates Real Execution

H4 is refuted. The fidelity ratio of TEA* is neither close to unity nor stable as the scenario tightens. Its missions sit systematically above the ideal diagonal in both layouts, with the deviation widening sharply in the hospital scenario, while the semaphore coordinator stays far closer to it (Figures 6.3 and 6.4; Sections 6.3.1 and 6.3.2). The hypothesis assumed that the richer temporal cost model would yield the more faithful execution estimate. The data invert that assumption: it is the reactive baseline, which carries no temporal model at all, whose planned cost tracks real execution most closely.

The divergence originates in how the planned cost is recorded. For each mission the logged planned cost is the free-flow traversal cost of a single route, the sum of the temporal costs of its edges recomputed from the static graph at every planning attempt, whereas the recorded execution time spans the whole mission, across every replanning cycle the supervisor triggers and the real blocking between them. The planned cost therefore describes a contention-free traversal of the route finally followed, while the execution time also absorbs the routes abandoned along the way. The ratio rises above one as a direct consequence, and widens the more a mission is replanned. The effect compounds in the hospital facility scenario, a robot whose own route is short but which spends most of the run waiting on contested nodes records a near-zero planned cost against a large execution time, producing the extreme ratios seen in the upper-left region of Figure 6.4. The semaphore behaves differently for a structural reason. It holds each robot until the edge ahead is free before allowing it to advance (Section 5.3), so its only divergence from the planned cost is the time spent blocked in the queues, which keeps its ratios far below those of TEA*.

This decoupling is a known phenomenon in literature: plan quality and execution fidelity separate as execution dynamics depart from the synchronous timing the plan assumes [25]. The general claim holds here, but its direction is the opposite of what the hypothesis drew from it. The system with the more sophisticated temporal model is the one that diverges most, precisely because the free-flow optimism of its logged per-route cost is what real execution violates. The cascaded waiting, supervisor interventions, and global replans documented as the shift from nominal path length to recovery overhead are exactly the costs absent from the logged planned cost [10], [19]. The semaphore avoids this divergence not because it predicts execution better, but because it only moves a robot once the edge ahead is free, so the planned and executed traversal end up coinciding.

An operator who reads the TEA* planned cost as an estimate of completion time will underestimate it systematically, by a margin that widens with structural contention. For scheduling and throughput decisions the KPI to trust is the measured execution time, not the planned cost, and the two should not be used as if they were the same. More broadly, plan fidelity has to be measured from a full run, by comparing the planned cost against the real execution time of the same mission. This benchmark was built to capture exactly that comparison.

7.6 Safety: Recoverable Planning Failures versus Irrecoverable Deadlock

H5 is partially confirmed. As predicted, the semaphore collapses at fleet sizes where TEA* still completes its runs, and it collapses earlier in the hospital scenario than in the industrial one (Figures 6.10 and 6.12). The hypothesis did not anticipate the second outcome: TEA* also fails at the largest fleets under the doubled workload, so its success rate is not perfect across the full range (Sections 6.5.1 and 6.5.2).

The two coordinators fail for fundamentally different reasons, and the difference matters for how safety is interpreted. The semaphore always produces a route, because spatial A* returns a path on any connected graph. Its failures are execution failures: two robots lock the edges each other needs, neither can advance, and the run ends in an irrecoverable deadlock (Section 5.3). TEA* can fail to find a valid plan when the space-time search exhausts its options under high fleet density and dense missions, but each such failure is recoverable: the supervisor re-invokes the planner and the fleet continues moving [10]. A run is only classified as `FAILED` when the accumulated planning failures reach the abort threshold, not at the first unsuccessful invocation. The semaphore's failures are therefore structurally irrecoverable; those of TEA* are not.

The semaphore enters deadlock early in the hospital scenario, where narrow corridors and single-access nodes eliminate the spatial detours that might otherwise absorb circular waits, and later in the open industrial layout where alternative routes remain available (Tables 6.1 and 6.2). TEA* reaches its planning limit only at high fleet sizes under the doubled workload, where the number of temporal constraints grows until no valid schedule can be found within the planner timeout. This is a known boundary in MAPF: even complete planners cannot guarantee a solution in bounded time under sufficient load [23].

Success rate is the correct safety KPI for comparing these two systems. Failed planning count is not a valid comparator when one system has no planner. Evaluated on success rate, TEA* maintains a wider operational envelope in both scenarios and both workloads, with its failures concentrated at the upper boundary of the tested range.

7.7 Methodological Contributions and Limitations

The preceding sections interpret what the two coordinators do. Those interpretations rest on a methodology and a body of software that are themselves an outcome of this dissertation, and that stand independently of any particular verdict on TEA* or the semaphore. This section separates the two: the methodological contributions on one side, and the boundaries within which the empirical conclusions hold on the other.

7.7.1 Methodological Contributions

The first contribution is the automated benchmarking pipeline (Section 5.1). It launches, monitors, and terminates every run under identical conditions, and the `kpi_collector` replaces the

original manual export step with an automatic one, so that a full campaign across coordinators, scenarios, fleet sizes, and workloads is executed without operator intervention. Coupled to it is the structured YAML schema, which turns each run into a self-contained, machine-readable record spanning fleet-level aggregates and per-mission detail. The analysis tool (Section 5.4) then aggregates dozens of these records into the five hypothesis-aligned views, closing the loop from execution to evidence without manual data handling.

A second contribution is the cross-layer comparison methodology. The benchmark treats the coordination stack as a single end-to-end evaluation unit and captures solution cost, execution time, replanning frequency, and plan fidelity jointly from the same run, rather than measuring any layer in isolation. The semaphore baseline (Section 5.3) completes the methodology by providing a reactive reference with no temporal look-ahead, against which the contribution of time-aware planning is isolated rather than merely asserted.

7.7.2 Limitations of the Study

The evaluation reports a single run per experimental condition (Chapter 6). This is sufficient to expose the qualitative regimes that dominate the results, the operational boundaries, the onset of deadlock, and the direction of the fidelity gap, but it yields point estimates without variance, and therefore does not support the mean $\pm$ standard-deviation summaries customary in comparative KPI reporting. Replication would matter most near the failure boundaries, where the outcome is a discrete success-or-collapse event and run-to-run sensitivity is highest.

Coordination pressure is induced structurally, through bottleneck topology and workload density, and not through explicitly injected execution disturbances such as timing slippage or communication dropout. This was a deliberate scope decision (Section 3.4.4), and it is enough to produce waiting, replanning, and deadlock in end-to-end runs. Its limit is that the disturbance models in the literature act directly on temporal desynchronisation [19], the same mechanism that drives the replanning overhead of H3 and the fidelity gap of H4; an evaluation that injected delays at controlled rates would probe the supervisor under conditions closer to the rationale for which it was designed [10].

All runs are simulated. Physical execution diverges from centrally computed plans even on real warehouse hardware, because executed action durations depart from the synchronous timing the plan assumes [25], and a field deployment would add localisation drift, mechanical slippage, and communication latency that the simulator does not reproduce, the very disturbances the TEA* supervisor was validated against in real environments [10]. The results therefore characterise relative coordinator behaviour under controlled structural contention, not absolute performance under field conditions.

Finally, the comparison spans two coordinators. The methodology is agnostic to the system under test: any coordinator that publishes on the standardised topics is evaluated through the same pipeline without modification (Section 5.3.3), and the two chosen here bracket the design space, from a look-ahead-free reactive baseline to a time-aware centralised planner. What the study does not include is the evaluation of intermediate designs or, more consequentially, of variants of

the INESC TEC coordinator itself. Measuring whether concrete implementation improvements to the TEA* stack, supervisor threshold calibration, pre-processor choke-point handling, or the scheduling policy, yield measurable KPI gains is the practical purpose this benchmark was built to serve, and it is the principal direction left to future work.

Chapter 8

Conclusions and Future Work

This chapter summarises the contributions of the work and outlines directions for future work.

8.1 Summary of Contributions

This dissertation set out to design and implement a benchmarking methodology capable of evaluating the INESC TEC fleet coordination system under realistic intralogistics conditions. The work delivers five contributions across system implementation, data collection, and experimental evaluation.

The first contribution is an automated ROS benchmarking pipeline (Section 5.1) that launches, monitors, and terminates every run under identical conditions while controlling the experimental parameters, scenario, fleet size, and workload. By removing the operator from the execution loop, it eliminates a source of measurement error and makes campaigns of dozens of runs reproducible without manual intervention. The second contribution is the structured YAML schema and the `mission_benchmark` node (Sections 5.2 and 5.1.3), which turn each run into a self-contained, machine-readable record of cross-layer KPIs.

The third contribution is the semaphore baseline (Section 5.3), a reactive reference coordinator developed specifically for this dissertation. It provides the controlled point of comparison against which the contribution of time-aware planning is isolated rather than asserted, and its behaviour exposes empirically the collapse of purely reactive coordination under increasing fleet density. The fourth contribution is the benchmarking analysis tool (Section 5.4), a system-agnostic visualisation environment that aggregates the YAML records into five views, each aligned with one of the scientific hypotheses of Section 4.3; because it depends only on the standardised output format and not on any particular coordinator, it remains reusable by the INESC TEC team across future iterations of the TEA* stack.

The fifth contribution is empirical: the quantitative characterisation of TEA* against a reactive baseline under systematic variation of fleet size N , workload, and bottleneck structure (Chapter 6),

framed as the explicit evaluation of five falsifiable hypotheses rather than as an open-ended comparison. Of the five hypotheses, two are confirmed, one is partially confirmed, one is partially refuted, and one is refuted. Across all cases, the pipeline produced the cross-layer evidence needed to reach a verdict, and the results show that TEA* and the semaphore differ primarily in whether the fleet can operate at all under structural contention, not in how efficiently it does so.

8.2 Future Work

The limitations identified in Section 7.7 and the reusable nature of the benchmarking artefacts together define the directions in which this work can be extended.

A first direction is the validation of the methodology on physical hardware. Repeating the benchmarking protocol on the AMR fleet at the INESC TEC iilab would quantify the gap between simulated and real coordinator behaviour, the localisation drift, mechanical slippage, and communication latency that the simulator does not reproduce and that Section 7.7 identifies as the principal threat to external validity. Because the `mission_benchmark` subscribes only to the standardised ROS topics shared by every coordinator (Section 5.2), a physical fleet that publishes on those same topics is measured by the existing pipeline without modification.

A second direction, and the most immediate application, is the comparison of successive versions of the TEA* coordinator. The analysis tool was deliberately designed to be system-agnostic (Section 5.4): it depends only on the standardised output format and not on any particular coordinator, so iterations of the INESC TEC stack, whether in supervisor threshold calibration, pre-processor choke-point handling, or revised scheduling policies, can be evaluated against one another through the same pipeline without any change to the measurement infrastructure (Section 5.3.3).

A third direction addresses the statistical scope of the evaluation. The campaigns reported here execute a single run per experimental condition, which yields point estimates without variance and does not support the mean $\pm$ standard-deviation reporting customary in comparative KPI studies, a limitation stated explicitly in Section 7.7. Executing ten or more runs per condition would produce confidence intervals and enable formal hypothesis testing, and the existing infrastructure already accommodates this: the pipeline launches any number of runs per condition (Section 5.1) and the analysis tool aggregates them automatically across the loaded dataset (Section 5.4), with replication most informative near the failure boundaries where the outcome is a discrete success-or-collapse event.

A fourth direction is the construction of additional test scenarios with controlled structural characteristics. The two facilities evaluated here, an open industrial layout and a constrained hospital one, differ along several structural axes at once, corridor width, single-access vertices density, and route redundancy, so the contribution of each cannot be isolated from the others. Building graph topologies that vary one such factor at a time would separate these effects, and because every scenario is defined as a topological graph ingested by the pipeline and selectable

through the scenario filter of the analysis tool (Sections 5.1 and 5.4), new scenarios extend the study without any change to the coordinators or the measurement code.

References

- [1] P. R. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses,” *AI magazine*, vol. 29, no. 1, pp. 9–20, 2008.
- [2] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, “Lifelong multi-agent path finding in large-scale warehouses,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 11 272–11 281.
- [3] Z. Yan, N. Jouandeau, and A. A. Cherrier, “A survey and analysis of multi-robot coordination,” *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.
- [4] N. Jouandeau and Z. Yan, “Decentralized waypoint-based multi-robot coordination,” in *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems*, IEEE, 2012, pp. 155–159. DOI: [10.1109/CYBER.2012.6392549](https://doi.org/10.1109/CYBER.2012.6392549).
- [5] Athira K. A., Divya Udayan J., and U. Subramaniam, “A systematic literature review on multi-robot task allocation,” *ACM Computing Surveys*, vol. 57, no. 3, 68:1–68:28, 2024. DOI: [10.1145/3700591](https://doi.org/10.1145/3700591).
- [6] 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*, vol. 23, no. 9, pp. 939–954, 2004.
- [7] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 2012, pp. 563–569.
- [8] J. Yu and S. M. LaValle, “Optimal multi-robot path planning on graphs: Structure and computational complexity,” *arXiv preprint arXiv:1507.03289*, 2015.
- [9] R. Stern, N. R. Sturtevant, A. Felner, *et al.*, “Multi-agent pathfinding: Definitions, variants, and benchmarks,” in *Proceedings of the International Symposium on Combinatorial Search (SoCS)*, vol. 10, 2019, pp. 151–158.
- [10] D. M. Matos, P. Costa, H. Sobreira, A. Valente, and J. Lima, “Efficient multi-robot path planning in real environments: A centralized coordination system,” *International Journal of Intelligent Robotics and Applications*, 2024. DOI: [10.1007/s41315-024-00378-3](https://doi.org/10.1007/s41315-024-00378-3).
- [11] S. Schaefer, L. Palmieri, L. Heuer, R. Dillmann, S. Koenig, and A. Kleiner, “A benchmark for multi-robot planning in realistic, complex and cluttered environments,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9231–9237.
- [12] E. Heiden, L. Palmieri, L. Bruns, K. O. Arras, G. S. Sukhatme, and S. Koenig, “Bench-mr: A motion planning benchmark for wheeled mobile robots,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4536–4543, 2021.

- [13] A.-I. Toma, H.-Y. Hsueh, H. A. Jaafar, R. Murai, P. H. J. Kelly, and S. Saeedi, "Pathbench: A benchmarking platform for classical and learned path planning algorithms," in *2021 18th Conference on Robots and Vision (CRV)*, IEEE, 2021.
- [14] D. Portugal and R. P. Rocha, "A ros-based framework for simulation and benchmarking of multi-robot patrolling algorithms," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 2501–2506.
- [15] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.
- [16] M. Čáp, P. Novák, A. Kleiner, and M. Selecký, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 935–949, 2015. DOI: [10.1109/TASE.2015.2445780](https://doi.org/10.1109/TASE.2015.2445780).
- [17] P. Moura, P. Costa, J. Lima, and P. Costa, "A temporal optimization applied to time enhanced a*," in *AIP Conference Proceedings*, vol. 2116, AIP Publishing, 2019, p. 220 007. DOI: [10.1063/1.5114225](https://doi.org/10.1063/1.5114225).
- [18] M. Berndt, D. Krummacker, C. Fischer, and H. D. Schotten, "Centralized robotic fleet coordination and control," in *Proceedings of the 25th ITG-Symposium on Mobile Communications*, Osnabrück, Germany, Nov. 2021.
- [19] D. Atzmon, R. Stern, A. Felner, G. Wagner, R. Barták, and N.-F. Zhou, "Robust multi-agent path finding and executing," *Journal of Artificial Intelligence Research*, vol. 67, pp. 549–579, 2020.
- [20] N. van Duijkeren, L. Palmieri, R. Lange, and A. Kleiner, *An industrial perspective on multi-agent decision making for interoperable robot navigation following the VDA5050 standard*, arXiv preprint arXiv:2311.14615, 2023. arXiv: [2311.14615](https://arxiv.org/abs/2311.14615) [cs.RO].
- [21] N. Kalra, R. Zlot, M. B. Dias, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.
- [22] E. Boyarski, A. Felner, R. Stern, *et al.*, "Icbs: Improved conflict-based search algorithm for multi-agent pathfinding," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 740–746.
- [23] A. Felner, R. Stern, S. E. Shimony, *et al.*, "Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges," in *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS)*, AAAI Press, 2017, pp. 29–37.
- [24] H. Ma, D. Harabor, P. J. Stuckey, J. Li, and S. Koenig, "Searching with consistent prioritization for multi-agent path finding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7643–7650.
- [25] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1125–1131, 2019. DOI: [10.1109/LRA.2019.2894217](https://doi.org/10.1109/LRA.2019.2894217).
- [26] L. Kästner, T. Bhuiyan, T. A. Le, *et al.*, "Arena-bench: A benchmarking suite for obstacle avoidance approaches in highly dynamic environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9477–9484, 2022. DOI: [10.1109/LRA.2022.3190086](https://doi.org/10.1109/LRA.2022.3190086).

Appendix A

Supplementary Data

A.1 Real-Time Monitoring Interface

The automated benchmarking pipeline includes a real-time KPI dashboard implemented as an rqt visualisation plugin. This interface subscribes to the metrics derived by the `mission_benchmark` node and provides live monitoring of the fleet's state, tracking mission progress, execution times, and replanning events as the simulation unfolds. Figure A.1 illustrates a snapshot of this dashboard during an active run.

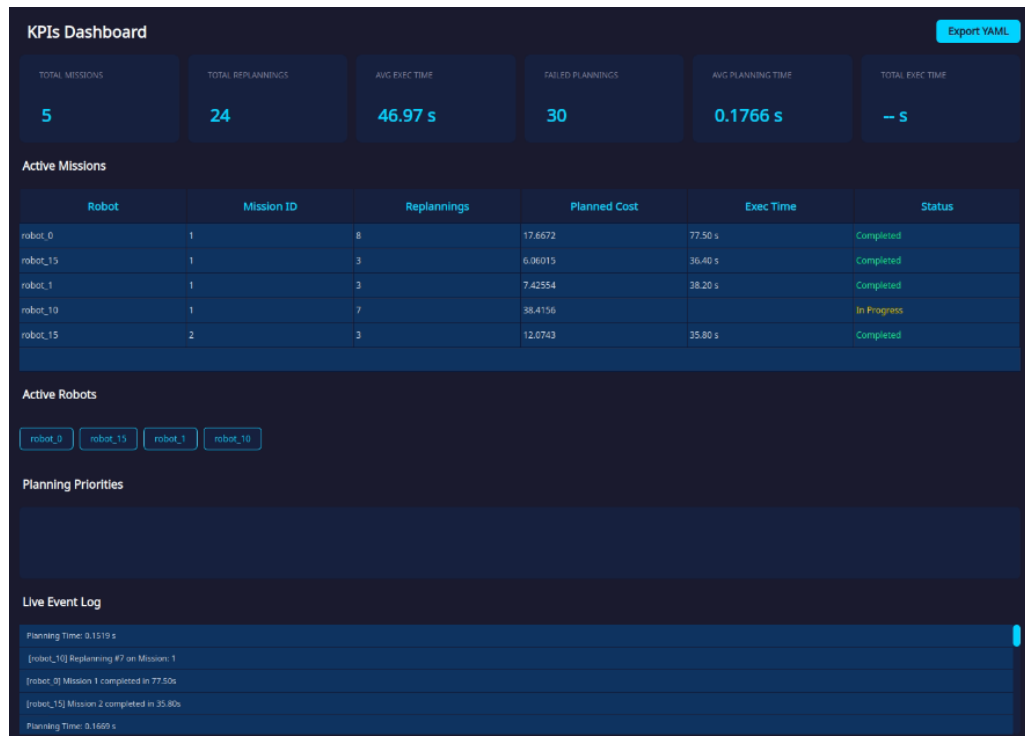


Figure A.1: Real-time KPI dashboard active during a run.

A.2 Analysis Tool

The following figure shows the Loader screen of the Detailed Analysis mode.

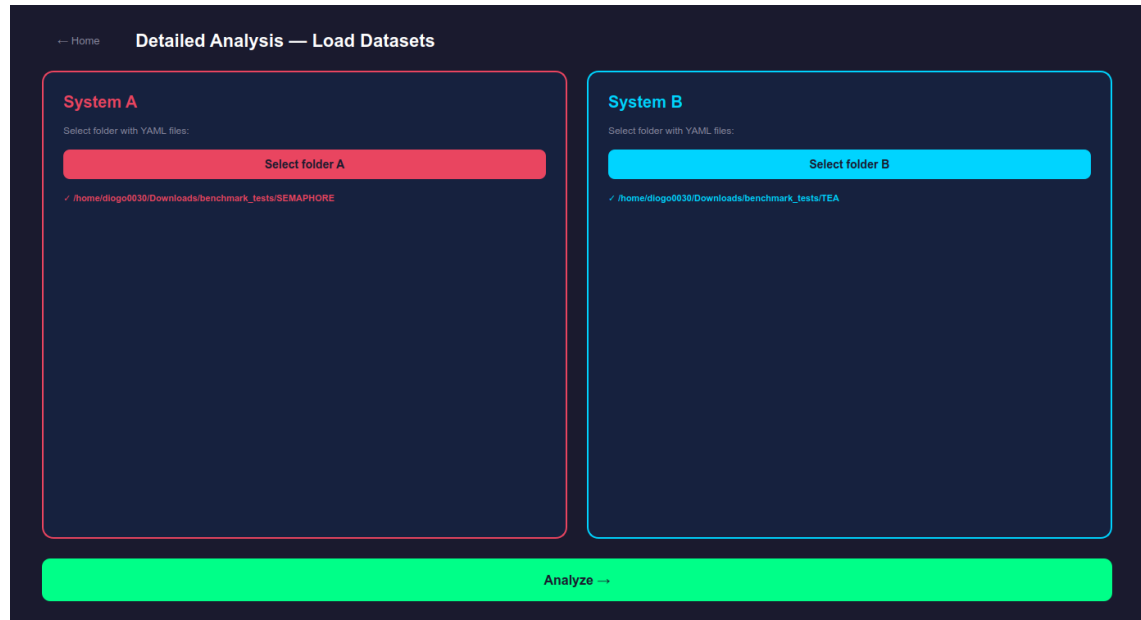


Figure A.2: Loader screen of the Detailed Analysis mode, with two folder pickers for loading datasets.

A.3 Example Benchmark Output (YAML)

Listing A.1 reproduces a complete YAML file produced by the `kpi_collector` at the end of a TEA* run, illustrating the three-block structure (`benchmark_meta`, `global_kpis`, and `robots`) described in Section 5.1.3. The file corresponds to a successful four-robot run in the industrial facility scenario under the single-task workload. Within the `robots` block, each robot records its per-mission `final_planned_cost`, `exec_time_s`, and `total_replannings`, together with the full `plan_history` listing the planned cost computed at every replanning attempt.

```

1 benchmark_meta:
2   scenario: "sim1_portic"
3   fleet_size: 4
4   test_status: "SUCCESS"
5
6 global_kpis:
7   total_missions: 4
8   total_replannings: 13
9   avg_exec_time_s: 75.15
10  total_failed_plannings: 30
11  avg_planning_time_s: 0.1758
12  total_execution_time_s: 122.20
13

```

```
14 robots:
15   robot_0:
16     total_missions: 1
17     missions_completed: 1
18     total_replannings: 5
19     avg_exec_time_s: 77.50
20     mission_details:
21       mission_1:
22         status: Completed
23         final_planned_cost: 29.507
24         total_replannings: 5
25         exec_time_s: 77.5
26         plan_history:
27           attempt_0:
28             cost: 77.1736
29           attempt_1:
30             cost: 77.1736
31           attempt_2:
32             cost: 77.1736
33           attempt_3:
34             cost: 43.8998
35           attempt_4:
36             cost: 43.8998
37           attempt_5:
38             cost: 29.507
39   robot_1:
40     total_missions: 1
41     missions_completed: 1
42     total_replannings: 2
43     avg_exec_time_s: 67.90
44     mission_details:
45       mission_1:
46         status: Completed
47         final_planned_cost: 27.2574
48         total_replannings: 2
49         exec_time_s: 67.9
50         plan_history:
51           attempt_0:
52             cost: 34.683
53           attempt_1:
54             cost: 34.683
55           attempt_2:
56             cost: 27.2574
57   robot_10:
58     total_missions: 1
59     missions_completed: 1
60     total_replannings: 3
61     avg_exec_time_s: 118.80
62     mission_details:
```

```

63     mission_1:
64         status: Completed
65         final_planned_cost: 43.5685
66         total_replannings: 3
67         exec_time_s: 118.8
68     plan_history:
69         attempt_0:
70             cost: 19.9008
71         attempt_1:
72             cost: 19.9008
73         attempt_2:
74             cost: 11.8025
75         attempt_3:
76             cost: 43.5685
77 robot_15:
78     total_missions: 1
79     missions_completed: 1
80     total_replannings: 3
81     avg_exec_time_s: 36.40
82     mission_details:
83         mission_1:
84             status: Completed
85             final_planned_cost: 6.06015
86             total_replannings: 3
87             exec_time_s: 36.4
88         plan_history:
89             attempt_0:
90                 cost: 36.3557
91             attempt_1:
92                 cost: 36.3557
93             attempt_2:
94                 cost: 6.06015
95             attempt_3:
96                 cost: 6.06015

```

Listing A.1: Complete TEA* benchmark output file (`sim1_portic, m1, N = 4`).

A.4 Plan Fidelity Ratio Tables

The following tables report the per-robot fidelity ratio for all robots, across both test scenarios.

Robot	System A Avg Cost	System A Avg Exec	System B Avg Cost	System B Avg Exec	System A Ratio	System B Ratio
robot_0	78.38	86.2	67.23	91.4	×1.10	×1.36
robot_1	34.68	50.9	34.68	57.3	×1.47	×1.65
robot_2	15.00	17.0	15.00	37.4	×1.13	×2.49
robot_3	81.04	89.1	54.79	123.4	×1.10	×2.25
robot_4	32.90	44.0	32.90	44.5	×1.34	×1.35
robot_5	50.07	52.9	49.25	67.6	×1.06	×1.37
robot_6	67.45	73.8	67.45	87.9	×1.09	×1.30
robot_7	47.92	49.2	32.98	64.5	×1.03	×1.96
robot_8	33.39	35.6	33.39	53.4	×1.07	×1.60
robot_9	76.95	81.9	60.91	101.5	×1.06	×1.67
robot_10	69.69	75.2	26.25	109.2	×1.08	×4.16
robot_11	12.29	12.3	12.29	25.4	×1.00	×2.06
robot_12	30.50	32.3	30.50	48.6	×1.06	×1.59
robot_13	37.27	37.8	36.11	54.8	×1.02	×1.52
robot_14	37.40	37.0	28.33	64.6	×0.99	×2.28
robot_15	36.36	37.9	36.36	54.0	×1.04	×1.49
robot_16	35.19	36.8	31.54	50.9	×1.04	×1.61
robot_17	16.19	16.6	16.19	27.1	×1.03	×1.67
robot_18	-	-	18.27	91.8	-	×5.03
robot_19	64.63	69.2	80.77	101.5	×1.07	×1.26

Figure A.3: Fidelity ratio per robot, industrial facility scenario. System A: semaphore coordinator, System B: TEA*.

Robot	System A Avg Cost	System A Avg Exec	System B Avg Cost	System B Avg Exec	System A Ratio	System B Ratio
robot_0	41.94	73.4	32.52	61.4	×1.75	×1.89
robot_1	47.24	84.5	6.94	76.4	×1.79	×11.02
robot_2	-	-	6.11	90.4	-	×14.80
robot_3	-	-	9.91	87.8	-	×8.85
robot_4	-	-	33.44	71.9	-	×2.15
robot_5	-	-	6.96	88.7	-	×12.74
robot_6	-	-	35.00	76.4	-	×2.18
robot_7	-	-	7.35	140.0	-	×19.04

Figure A.4: Fidelity ratio per robot, hospital facility scenario. System A: semaphore coordinator, System B: TEA*.

A.5 Replanning Detail Tables

The following figures report the average replanning events per robot for both coordinators, as produced by the benchmarking analysis tool.

Robot	System A Avg Replans	System B Avg Replans	Diff %
robot_0	0.00	12.21	+100.0%
robot_1	0.00	7.21	+100.0%
robot_2	0.00	3.92	+100.0%
robot_3	0.00	17.25	+100.0%
robot_4	0.00	11.67	+100.0%
robot_5	0.00	12.45	+100.0%
robot_6	0.00	14.09	+100.0%
robot_7	0.00	13.80	+100.0%
robot_8	0.00	9.56	+100.0%
robot_9	0.00	18.00	+100.0%
robot_10	0.00	11.44	+100.0%
robot_11	0.00	3.67	+100.0%
robot_12	0.00	8.44	+100.0%
robot_13	0.00	10.88	+100.0%
robot_14	0.00	11.33	+100.0%
robot_15	0.00	8.71	+100.0%
robot_16	0.00	9.83	+100.0%
robot_17	0.00	5.33	+100.0%
robot_18	0.00	19.67	+100.0%
robot_19	0.00	14.09	+100.0%

Figure A.5: Average replanning events per robot, industrial facility scenario. System A: semaphore coordinator, System B: TEA*.

Robot	System A Avg Replans	System B Avg Replans	Diff %
robot_0	0.00	13.70	+100.0%
robot_1	0.00	17.44	+100.0%
robot_2	0.00	18.87	+100.0%
robot_3	0.00	17.23	+100.0%
robot_4	0.00	12.17	+100.0%
robot_5	0.00	15.71	+100.0%
robot_6	0.00	13.80	+100.0%
robot_7	0.00	13.33	+100.0%

Figure A.6: Average replanning events per robot, hospital facility scenario. System A: semaphore coordinator, System B: TEA*.