University of Gloucestershire

Employing a distributed auction algorithm in decentralised mobile manipulators for independent and collaborative assembly tasks

Submitted in partial fulfilment of the

requirements for the degree of Bachelor of

Mechatronics Engineering

Tom Le Huray
5-24-2025
S4220534

I. ABSTRACT

This dissertation investigates distributed auction algorithms for decentralised control of dual mobile manipulators in collaborative assembly tasks. Traditional multi-robot systems rely on centralised architectures that introduce single points of failure and limit scalability. This research extends Zavlanos' theoretical framework with enhancements for consensus maintenance under communication constraints, failure detection and recovery, and collaborative task handling. The implementation achieves provable convergence properties, bounded optimality gaps, and formal completeness guarantees for recovery operations.

Experimental evaluation demonstrates the algorithm's performance across variations in task count, communication delay, packet loss probability, and minimum bid increment. Results show the algorithm maintains functionality under adverse communication conditions while achieving performance ratios of 0.72-0.85 relative to optimal solutions. Statistical analysis reveals task count as the dominant performance factor, while the time-weighted consensus protocol effectively handles communication delays.

Implementation challenges were identified in recovery mechanism functionality and optimality guarantee relationships. This research contributes a foundation for decentralised control strategies in industrial mobile manipulation, providing mathematical guarantees while acknowledging practical implementation challenges.

II. EXECUTIVE SUMMARY

This research addresses a fundamental challenge in multi-robot systems: developing decentralised control algorithms with mathematical guarantees that maintain performance under realistic constraints. Traditional centralised approaches introduce single points of failure and limit scalability, while current decentralised methods often lack formal guarantees or struggle with physical implementation constraints.

The project develops a distributed auction algorithm for dual mobile manipulators performing collaborative assembly tasks, extending Zavlanos' theoretical framework to incorporate robust communication protocols, failure recovery mechanisms, and collaborative task handling. The primary contributions include:

- A mathematically rigorous distributed auction algorithm with provable convergence time bound of $O(K^2 \cdot b_{max}/\varepsilon)$ and optimality gap of 2ε, where K is the number of tasks, b_{max} is the maximum bid value, and ε is the minimum bid increment.
- A time-weighted consensus protocol with exponential convergence properties that maintains system-wide consistency despite communication delays and packet loss.
- A dual-mechanism approach to failure detection combining heartbeat monitoring and progress tracking, with formal recovery guarantees.
- Comprehensive performance evaluation across systematically varied parameters, demonstrating algorithm robustness to communication constraints.

The methodology combined mathematical analysis with computational validation, initially planned as a dual-phase approach with MATLAB development followed by ROS2/Gazebo simulation. Due to implementation challenges with the ROS/Gazebo platform, the research focused exclusively on MATLAB and Python implementations for algorithm verification.

Experimental results demonstrate successful task allocation with workload balancing across robots, maintaining functionality under communication delays up to 500ms and packet loss rates up to 50%. However, several limitations were identified: (1) recovery mechanism issues preventing proper failure detection, (2) premature convergence with unassigned tasks, and (3) inconsistencies between theoretical ε effects and observed behaviour.

Statistical analysis revealed that task count predominantly impacts system performance metrics, with a near-linear relationship to message count ($R^2 = 0.976$) and significant effects on makespan and optimality gap. The algorithm achieved a performance ratio of 0.85 relative to

optimal solutions for small problem instances, degrading to approximately 0.72 for larger instances.

This research contributes both theoretical advancement and practical implementation insights to the field of decentralised multi-robot control, establishing a foundation for future development of robust, distributed coordination systems for industrial assembly applications. Recommended future work includes recovery mechanism redesign, convergence criteria enhancement, parameter auto-tuning, and validation on physical hardware platforms.

III. DECLARATION OF AUTHORSHIP

The work described in this report is the result of my own investigations. All sections of the text and results that have been obtained from other work are fully referenced. I understand that cheating and plagiarism constitute a breach of University Regulations and will be dealt with accordingly.

Signed:  Date: 24-05-2025

IV. ACKNOWLEDGEMENTS

First and foremost, I would like to thank my dissertation supervisors Dr Muhammad Babar Rasheed and Professor Shujun Zhang, whose guidance and feedback have been invaluable throughout this project. I would also like to thank my personal tutor David Plummer for his guidance and input during the early stages of the project, helping to narrow my focus and maintain a reasonable scope. Finally, I would like to thank my friends and family for their support and valuable input in helping me ensure the quality of my work.

V. TABLE OF CONTENTS

VI. TABLES

VII. TABLE OF FIGURES

VIII. TABLE OF ABBREVIATIONS

Table 1 Table of Abbreviations

Abbreviation	Definition
ADMM	Alternating Direction Method of Multipliers
ANOVA	Analysis of Variance
ARO	Autonomous Robotic Organisation
CPU	Central Processing Unit
CPM	Critical Path Method
DAG	Directed Acyclic Graph
DOF	Degrees of Freedom
DQN	Deep Q-Network
F/T	Force/Torque
GPU	Graphics Processing Unit
IMU	Inertial Measurement Unit
MATLAB	Matrix Laboratory
ODE	Open Dynamics Engine
RGB-D	Red Green Blue Depth
ROS	Robot Operating System
URDF	Unified Robot Description Format

IX. LIST OF SYMBOLS

Symbol	Definition
ε	Epsilon (minimum bid increment)
K	Number of tasks
b_{max}	Maximum bid value
$O(K^2)$	Big O notation for computational complexity
μ	Mu (convergence rate parameter)
λ	Lambda (information decay rate)
γ	Gamma (consensus weight factor)
τ	Tau (communication delay)
β	Beta (recovery auction weights)
p_{loss}	Probability of packet loss
T^f	Tasks to be assigned as failed

1 INTRODUCTION

1.1 RESEARCH CONTEXT

Through the rapid expansion and evolution of industrial automation, the world has seen a dramatic shift from isolated robotic systems toward coordinated multi-robot solutions that enhance a system's flexibility, reliability, and scalability (Skaugset, de Sousa and Sørensen, 2025). In spite of these advancements, contemporary multi-robot systems predominantly rely on centralised control architectures that create critical vulnerabilities through single points of failure and impose fundamental limitations on a system's scalability and reactivity (Olfati-Saber, Fax and Murray, 2007; Shorinwa et al., 2023) Decentralised control architectures represent a significant advancement. They distribute decision-making across individual agents, thus improving system robustness against node failures—a particularly valuable feature in bandwidth-constrained environments (Rossi et al., 2021).

Recent theoretical advances in consensus algorithms and high-precision localisation have substantially improved multi-robot coordination in controlled laboratory environments (Talamali *et al.*, 2021; Wang *et al.*, 2021). These advancements, however, face formidable implementation challenges when applied to real-world assembly tasks requiring precise coordination and real-time adaptation (Kim *et al.*, 2022). Whilst manufacturing sectors have adopted robotics for structured operations successfully, other industries—especially those with collaborative assembly—face distinctive challenges due to dynamic environments, communication constraints and strict synchronisation requirements (De Soto *et al.*, 2019; Bajracharya *et al.*, 2023). The integration of market-based mechanisms—principally distributed auction algorithms—presents a mathematically rigorous yet computationally efficient approach to decentralised task allocation that can achieve near-optimal solutions under realistic constraints (Bertsekas, 1988; Zavlanos, Spesivtsev and Pappas, 2008).

1.2 PROBLEM STATEMENT

Despite substantial theoretical advances being made in consensus protocols, distributed optimisation, and multi-agent coordination, a significant gap persists between the mathematical theory and practical implementation of decentralised control for mobile manipulators. Current research has established robust theoretical frameworks for distributed decision-making (Olfati-Saber, Fax and Murray, 2007; Shorinwa *et al.*, 2023), yet these frameworks frequently rely on

simplifying assumptions that neglect critical real-world constraints, including communication limitations, task dependencies, temporal constraints, and agent failures.

Three interlaced technical challenges constitute the core problem area of the research. First, existing solutions demonstrate inadequate real-time adaptation capabilities during physical interactions between robots in unstructured environments. Sheng *et al.* (2021) empirically established that conventional mobile manipulators require extensive calibration procedures to achieve even modest positioning accuracy, exhibiting mean errors of 4.7cm without specialised calibration. Second, while auction-based task allocation demonstrates compelling theoretical properties, Xiong *et al.* (2023) quantitatively demonstrated that practical implementations face fundamental trade-offs between computational efficiency and optimisation quality, with optimal task allocation becoming computationally intractable as system complexity increases. Third, Zhang *et al.* (2020) analytically verified that decentralised control systems often struggle to maintain stable cooperative manipulation under realistic constraints, documenting performance degradation of up to 37% when communication limitations impact system performance.

This research attempts to address these challenges by extending a mathematical formulation of a distributed auction algorithm (Zavlanos, Spesivtsev and Pappas, 2008) with provable convergence guarantees, robustness to communication limitations, and formal completeness properties for failure recovery, implemented within a high-fidelity simulation environment that incorporates realistic constraints and disturbances.

1.3 Project Scope and Limitations

The dissertation focuses on the theoretical foundations, algorithmic implementation, and performance characteristics of a novel distributed auction algorithm for decentralised control of dual mobile manipulators performing complex assembly tasks. The project encompasses the rigorous mathematical formulation of auction mechanisms with provable guarantees, implementation of resilient consensus protocols, development of robust failure detection and recovery strategies, and comprehensive performance evaluation within a high-fidelity simulation environment.

The project scope explicitly includes:

1. Formal mathematical formulation and analysis of a distributed auction algorithm with provable convergence properties, bounded optimality gaps, and quantifiable performance guarantees

2. Implementation of delay-tolerant communication protocols and time-weighted consensus mechanisms designed to maintain system-wide consistency despite realistic communication constraints
3. Development of probabilistic failure detection and deterministic recovery strategies with formal completeness guarantees and bounded recovery times
4. Validation through systematically designed simulated assembly scenarios with parameterised complexity levels and precisely controlled constraint variations
5. Rigorous comparative performance evaluation against centralised optimal allocation, greedy heuristics, and state-of-the-art decentralised approaches using standardised metrics

This research deliberately excludes physical hardware implementation, human-robot interaction scenarios, economic feasibility analysis, and extension to more than two robots to maintain a focused investigation of the foundational algorithmic and theoretical principles. These carefully considered limitations enable the in-depth examination of the mathematical guarantees and the performance characteristics that constitute the core scientific contribution of the work.

1.4 Project Objectives

The central aim of this work is to establish how a mathematically rigorous distributed auction algorithm can enable decentralised control in dual mobile manipulators to achieve provably robust and efficient collaborative assembly tasks without central coordination. The investigation is structured around three interconnected technical objectives with explicitly defined success criteria:

1.4.1 Objective 1: Distributed Auction Algorithm Development and Analysis

Develop and mathematically analyse a novel distributed auction algorithm with formally proven convergence properties, quantifiable optimality bounds, and demonstrated robustness to communication constraints. Extending Zavlanos, Spesivtsev and Pappas' (2008) theoretical framework, this objective establishes rigorous mathematical foundations for the algorithm's performance, providing theoretical guarantees, including convergence time complexity of $O(K^2 \cdot b_{max}/\varepsilon)$ and bounded by an optimality gap of 2ε, where K is the number of tasks, $b_{\max}$ is the maximum bid value, and ε is the minimum bid increment.

1.4.2 Objective 2: Consensus Protocol and Failure Recovery Mechanisms

Design and implement time-weighted consensus protocols that maintain system-wide consistency despite communication limitations, alongside probabilistic failure detection and deterministic recovery mechanisms with completeness guarantees. Building upon Zhang *et al.'s* (2020) findings on hardware constraints and Talamali *et al.'s* (2021) work on constrained communication; this objective develops and analyses recovery mechanisms with proven bounded recovery time and makespan degradation properties.

1.4.3 Objective 3: High-Fidelity Simulation Validation and Comparative Analysis

Develop a comprehensive simulation environment incorporating realistic constraints to evaluate the algorithm across systematically varied scenarios, validating theoretical guarantees and quantifying performance advantages compared to alternative approaches. Inspired by Xiong *et al.*'s (2023) selective planning framework, this objective delivers statistically rigorous performance analysis across standardised metrics including makespan, communication overhead, and robustness to failures, with direct comparison to centralised optimal allocation and alternative decentralised approaches.

2 LITERATURE REVIEW

2.1 THEORETICAL FOUNDATIONS OF DISTRIBUTED DECISION-MAKING IN MULTI-AGENT SYSTEMS

2.1.1 Optimisation Theory In Distributed Systems

Distributed optimisation frameworks have evolved significantly since Bertsekas' (1988) pioneering auction algorithm, which demonstrated how distributing computation across agents could efficiently solve complex assignment problems with $O(N^2A)$ complexity – a significant improvement over $O(N^3)$ centralised approaches. Here, N represents the number of agents and tasks whilst A denotes the maximum benefit value of the assignment problem. Olfati-Saber, Fax and Murray (2007) established critical theoretical connections between network topology and convergence properties, proving that convergence rate directly correlates with the algebraic connectivity of the communication graph. Recent work by Shorinwa *et al.* (2023) categorises distributed methods into first-order, sequential convex, and ADMM (Alternating Direction Method of Multipliers) variants, revealing fundamental trade-offs between communication overhead and computational efficiency.

In these optimisation algorithms, the convergence rate is typically expressed using the parameter ϵ, which represents the desired accuracy or error of a solution. First-order methods demonstrate $O\left(\frac{1}{\epsilon}\right)$ convergence rates with minimal computation but require $O\left(\frac{1}{\epsilon}\right)$ communication rounds, while second-order approaches achieve $O(\log(\frac{1}{\epsilon}))$ convergence at the cost of increased computational complexity: critical considerations for resource-constrained mobile robots operating in bandwidth-limited environments. The $O(1/\epsilon)$ notation indicates that approximately $1/\epsilon$ iterations are needed to achieve an error of at most ϵ, while $O(log(1/\epsilon))$ means only approximately $log(1/\epsilon)$ iterations are required, representing a substantially faster convergence, particularly for small error tolerances.

2.1.2 Game Theoretic Approaches To Multi-Agent Coordination

Game theory provides powerful structures for analysing strategic interaction in multi-agent systems. Rossi *et al.* (2021) systemically categorise market-based mechanisms, highlighting their moderate scalability and communication efficiency, with auction-based approaches achieving $O(N^2)$ communication complexity compared to $O(N^4)$ for centralised solutions. Zavlanos, Spesivtsev and Pappas (2008) leveraged competitive bidding principles to develop

their distributed auction algorithm, mathematically proving convergence to assignments within 2ε of optimal solutions, where ε represents the minimum bid increment. However, these formulations typically rely on idealised rational agents with perfect information—assumptions that rarely hold in physical systems with perception limitations, communication delays, and computational constraints. This creates a significant gap between theoretical guarantees and practical implementation that remains insufficiently addressed in the current literature.

2.1.3 Convergence And Optimality Guarantees In Distributed Algorithms

Establishing rigorous guarantees is crucial for reliable system performance. Nedić, Olshevsky and Rabbat (2018) provide fundamental convergence rate analyses for consensus-based optimisation, demonstrating that network topology directly impacts performance with explicit rates for various graph structures: $O(n^2 \log\left(\frac{1}{\epsilon}\right))$ iterations for path graphs versus $O(\log\left(\frac{1}{\epsilon}\right)$ for expander graphs. Zhang, Yang and Başar (2021) extend these insights to reinforcement learning contexts, proving that decentralised actor-critic methods can achieve asymptotic convergence under appropriate communication protocols. Notably, Talamali *et al.* (2021) demonstrate that constrained communication can paradoxically improve adaptation in specific multi-agent settings, with experiments showing 100% adaptation success with limited communication versus only 33% with global communication. These findings highlight a critical tension between theoretical optimality and practical implementation: theoretical guarantees typically assume idealised conditions rarely found in physical multi-robot systems, while practical implementations often sacrifice formal guarantees for operational robustness.

2.2 Evolution of Multi-Robot Coordination

2.2.1 Shifts In Multi-Robot Systems Design

Recent literature reveals a paradigm shift from isolated robotic systems toward coordinated multi-robot solutions. Skaugset, de Sousa and Sørensen (2025) propose transitioning from individual robots to Autonomous Robotic Organizations (AROs) for marine operations, directly addressing the vulnerabilities of centralised architectures identified by Olfati-Saber, Fax and Murray (2007) - specifically, their susceptibility to single points of failure. Empirically, Bone *et al.* (2023) demonstrate the performance advantages of decentralised coordination, achieving a 30% reduction in exploration times while maintaining resilience to communication constraints. However, their approach relies on homogeneous agents with simplified kinematic models, raising questions about applicability to heterogeneous teams of mobile manipulators with complex dynamics. These studies reveal a persistent methodological tension: theoretical

studies emphasise mathematical guarantees under idealised conditions, while implementation-focused research often sacrifices formal guarantees for practical functionality; creating a significant research opportunity for approaches that bridge this gap.

2.2.2 Consensus Mechanisms For Collaborative Decision-Making

Recent advances in consensus protocols have significantly improved multi-robot coordination capabilities. Wang *et al.* (2021) demonstrated that high-precision localisation could be achieved through modular approaches incorporating feedback loops between global and local consensus mechanisms, achieving sub-centimetre precision in complex environments. In stark contrast to conventional wisdom, Talamali *et al.* (2021) revealed that limiting communication can enhance collective adaptability to environmental changes, with their cross-inhibition pattern resolving conflicting information without deadlocks. These studies demonstrate a trend toward decentralised consensus mechanisms that balance precision and adaptability. However, significant gaps remain in developing protocols that offer formal convergence guarantees while operating under realistic constraints such as time-bounded decision-making and communication limitations. Current approaches typically optimise for either theoretical guarantees or practical robustness but rarely achieve both simultaneously—a critical limitation for mobile manipulators performing assembly tasks.

2.2.3 Emergent Behaviours In Multi-Robot Teams

Yang *et al.* (2019) developed a framework facilitating emergent coordination through self-reactive planning, using Behaviour Trees and Maslow-inspired priority mechanisms to enable dynamic adaptation without centralised control. Their experiments with 20 robots demonstrated quantifiable performance differences between priority schemes, with energy conservation varying by up to 15.42% between configurations. However, their approach lacks formal guarantees regarding convergence or optimality. This highlights a persistent methodological tension: approaches emphasising emergent behaviours typically offer adaptability but lack formal guarantees, whilst theoretically grounded approaches provide guarantees but often fail under real-world conditions—creating significant opportunities for research bridging this gap.

2.3 Market-Based Mechanisms for Decentralised Control

2.3.1 Distributed Auction Algorithms

Market-based mechanisms, particularly distributed auctions, offer a balance between theoretical rigour and computational efficiency. Zavlanos, Spesivtsev and Pappas (2008) introduced a distributed auction algorithm functioning with only local communication,

mathematically proving convergence to within 2ε of optimal solutions with time complexity bounded by $O(K^2 \cdot b_{max}/\varepsilon)$, where K represents tasks, and b_{max} the maximum bid value. More recently, Xiong *et al.* (2023) demonstrated that selective task planning can achieve 80% of optimisation objectives while using just 11% of computation time. However, both approaches exhibit significant limitations for physical implementation: Zavlanos assumes perfect communication and ignores physical constraints affecting bid valuation. At the same time, Xiong doesn't address how task criticality should be determined under uncertainty or how selective optimisation affects formal guarantees. These limitations create a critical research gap in developing auction mechanisms that maintain theoretical guarantees while accommodating physical constraints inherent in mobile manipulation.

2.3.2 Consensus Protocols Under Communication Restraints

The efficacy of auction mechanisms depends on underlying communication protocols. Olfati-Saber, Fax and Murray (2007) established that consensus convergence speed correlates with network algebraic connectivity, demonstrating performance deterioration when time delays exceed $\tau > \pi/2\lambda_n$. Contradicting conventional wisdom, Talamali *et al.* (2021) proved that constrained communication can enhance collective adaptation, achieving 100% adaptation success with limited communication versus only 33% with global communication. Silva *et al.* (2022) extended these insights with ACHORD, a coordination framework addressing intermittent connectivity that achieved reliable coordination despite network delays ranging from 70-173ms. These findings suggest optimal protocols should implement deliberate communication constraints rather than maximising information exchange. However, current literature lacks a unified framework integrating these insights with formal guarantees for distributed auction algorithms—a critical research gap for enabling robust decentralised control in mobile manipulators.

2.3.3 Robustness And Scalability Of Market-Based Approaches

Distributed auction algorithms demonstrate significant advantages for multi-robot task allocation. Shorinwa *et al.* (2023) emphasise that auction-based algorithms offer favourable trade-offs between computational complexity and solution quality, maintaining near-optimal performance as system complexity increases. Regarding scalability, Xiong *et al.* (2023) demonstrated that selective auction approaches maintain performance with bounded optimality gaps, scaling more favourably than centralised approaches for large task spaces. Research on explicit bounds for intermittent disconnections (Nunes *et al.*, 2017) and detection frameworks for malicious agents (Shibata *et al.*, 2023) provides mathematical guarantees under both denial-

of-service and deception attacks. However, most existing research examines these properties in isolation, without considering their interdependencies or collective impact on system performance—representing a significant research gap for mobile manipulators performing assembly tasks where robustness, scalability, and optimality must be balanced within a unified framework.

2.4 Kinematic and Control Challenges in Mobile Manipulation for Collaborative Assembly

2.4.1 Precision And Calibration In Mobile Manipulation

Achieving the high positional accuracy required for assembly tasks presents significant challenges for mobile manipulators. Bostelman, Hong and Marvel (2016) highlighted the lack of standardised metrics despite growing industrial relevance, creating difficulties for comparative evaluation. Empirically, Sheng *et al.* (2021) established that conventional mobile manipulators exhibit mean errors of 4.7cm without specialised calibration, with their multi-level measurement strategy reducing errors by 93% to achieve sub-millimetre accuracy. However, this approach requires centralised data processing and doesn't address calibration maintenance in distributed systems. Current literature largely treats calibration as separate from coordination algorithms—creating an artificial divide between theoretical approaches assuming perfect calibration and practical implementations that must continuously adapt to calibration drift and environmental uncertainties.

2.4.2 Dual-Arm Manipulation Strategies

Coordinating multiple manipulators requires specialised control strategies that significantly impact distributed coordination algorithms. Likar, Nemec and Žlajpah (2016) introduced the concept of virtual mechanisms, modelling tasks as kinematic chains between manipulators to enable coordinated motion with moving bases—a powerful abstraction for representing complex manipulation constraints. Kim *et al.*, (2022) quantitatively demonstrated that kinematic configuration critically impacts performance, with an Amiro2-ZYZ configuration achieving 34% higher performance in assembly tasks compared to alternatives. However, both approaches rely on centralised control assumptions and incomplete failure recovery mechanisms. These frameworks offer valuable task representation standards that could complement distributed auction algorithms by integrating kinematic constraints into bid valuations for decentralised assembly operations, though significant research gaps remain in developing approaches that combine kinematic optimisation with distributed coordination.

2.4.3 Motion Planning In Constrained Environments

Effective navigation within confined spaces requires sophisticated planning approaches that must integrate with distributed coordination. Delgado *et al.* (2023) demonstrated efficiency gains through coordinated base-arm movements, achieving 27% energy savings compared to decoupled planning. De Soto *et al.* (2019) examined the organisational impacts of automation, highlighting that effective implementation requires not only technical solutions but also organisational adaptations—a dimension often overlooked in algorithm-focused research. Current literature treats motion planning and organisational integration as separate from distributed coordination—creating an artificial divide between technical algorithms and implementation context, representing a significant research gap for distributed auction algorithms applied to mobile manipulators in industrial settings.

2.5 Communication Limitations and Failure Recovery in Multi-Robot Systems

2.5.1 Communication Models And Limitations In Multi-Robot Systems

Real-world communication introduces constraints that significantly impact distributed coordination. (Gielis *et al.*, 2022) revealed that 87% of papers rely on idealised communication assumptions that rarely hold in practice—a methodological limitation undermining the applicability of theoretical results. Silva *et al.* (2022) provided quantifiable performance metrics for intermittent connectivity, including communication ranges of 70-173m and variable network delays. (Wu, Zöcklein, and S. Brell-Çokcan, 2024) demonstrated 5G-enabled coordination achieving sub-50ms latencies across various protocols—a significant advancement for real-time coordination in bandwidth-intensive applications like collaborative assembly. Research gaps remain in cross-protocol integration, security-performance trade-offs, and standardised evaluation metrics, particularly for heterogeneous teams under bandwidth constraints. These limitations are especially relevant for distributed auction algorithms typically assuming reliable communication for bid exchange—representing a critical research opportunity for developing auction mechanisms maintaining performance guarantees despite unreliable communication.

2.5.2 Failure Detection And Recovery Mechanisms

Ensuring system resilience against component failures is critical for practical deployment. Zhang *et al.* (2020) quantified 37% performance degradation under communication failures in decentralised control, though their homogeneous DQN approach maintained 97.4% success rates despite disturbances. Abbas and Dwivedy, (2021) reduced network bandwidth consumption by 69% while maintaining operational integrity under packet loss rates of up to

40%, providing explicit stability guarantees for specific failure patterns. Zhang, Yang and Başar (2021) established mathematical bounds for tolerable attack parameters enabling consensus despite malicious interference, demonstrating that consensus can be maintained when attack frequency satisfies $\tau_a > \tau_0 + ln(\mu)/\lambda$. Prorok *et al.* (2021) distinguished between robustness (performance under known uncertainties) and resilience (adaptation to unpredicted disturbances), revealing that current research disproportionately emphasises pre-designed solutions rather than adaptive recovery mechanisms. These findings highlight a fundamental contradiction between communication efficiency and system security, with persistent gaps in physical validation and standardised resilience metrics—underscoring the need for recovery mechanisms with formal guarantees validated under realistic constraints.

2.5.3 Performance Evaluation Under Real-World Constraints

Quantifying system success requires appropriate metrics and realistic testing environments. Bajracharya *et al.* (2023) demonstrated that metrics-driven evaluation significantly improves mobile manipulation performance, achieving 35.7% reliability after six field tests through systematic tracking. Russo (2022) identified 28 distinct performance metrics across mobile manipulation studies with little consistency—creating significant challenges for comparative evaluation. Both studies highlight the necessity of objective performance indices balancing computational efficiency with reliability while accounting for real-world constraints. However, current literature lacks standardised benchmarks specifically for distributed coordination algorithms applied to mobile manipulators—a methodological gap limiting meaningful comparison between approaches and representing a critical research opportunity for developing integrated evaluation frameworks assessing both theoretical properties and practical performance.

2.6 Literature Review Conclusions

This review has identified critical research gaps at the intersection of distributed auction algorithms and mobile manipulation: (1) integration of theoretical guarantees with practical robustness, (2) adaptation to communication limitations, (3) incorporation of physical constraints, (4) unified failure recovery with formal guarantees, and (5) standardised evaluation methodologies. Addressing these gaps requires developing mathematically rigorous algorithms with provable properties that maintain performance under realistic constraints—bridging the persistent divide between theoretical formulations and practical implementations to advance decentralised control for mobile manipulators performing collaborative assembly tasks.

3 Methodology

3.1 Requirements and Specifications

Developing a decentralised control system for dual mobile manipulators requires careful consideration of theoretical foundations and practical implementation constraints. Building upon the mathematical framework established in the Literature Review, the system must satisfy several key requirements. These include algorithmic completeness to guarantee task allocation for all executable tasks, theoretical guarantees to maintain provable bounds on convergence time and optimality gap, communication resilience to function under realistic constraints, failure recovery to detect and recover from individual robot failures, and real-time performance to ensure computational feasibility.

The complete technical specifications for the system are documented in Appendix A, which defines quantifiable requirements across system architecture, robot specifications, algorithm properties, communication parameters, and performance metrics. Key specifications include maintaining the theoretical convergence bound of $O(K^2 \cdot b_{\max}/\varepsilon)$, operating successfully under communication delays up to 500ms and packet loss up to 50%, and achieving assembly precision of ±0.5mm in simulation.

3.2 System Architecture

The system architecture follows a modular design with distinct functional components organised hierarchically, as illustrated in Figure 1. Each robot maintains its own controller with components for task management, auction processing, consensus maintenance, failure detection, and execution control. The communication layer implements a peer-to-peer protocol

with asynchronous message passing for auction announcements, time-weighted information updating for consensus maintenance, and heartbeat signals for failure detection.

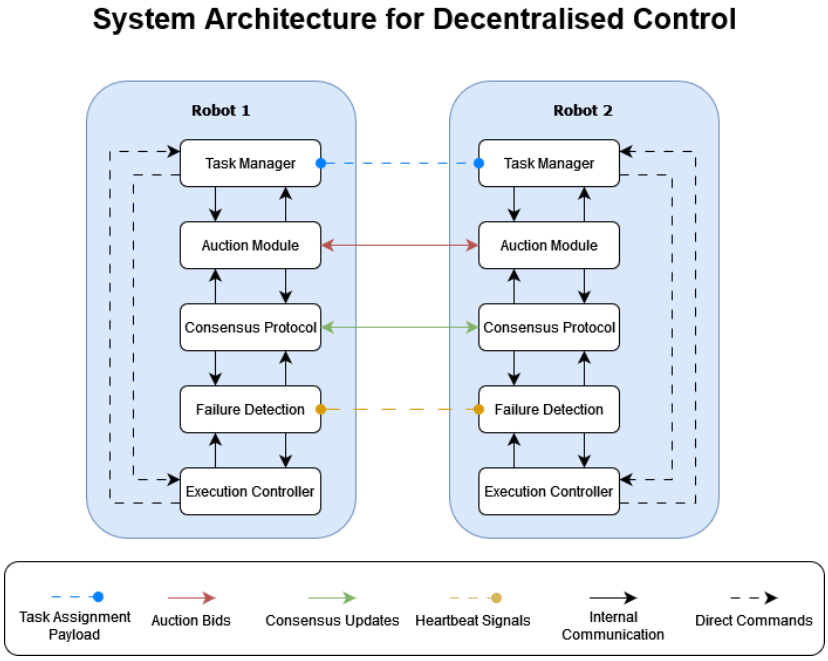

Figure 1 System Architecture for Decentralised Control of Dual Mobile Manipulators. The diagram illustrates inter-robot communication (horizontal connections) and intra-robot information flow (vertical connections). Inter-robot communication includes auction bids (red arrows), consensus updates (green arrows), and heartbeat signals (orange dashed lines). Within each robot, modules communicate through bidirectional connections for state updates and commands, with direct command paths between the Task Manager and Execution Controller. This fully decentralised architecture eliminates the need for central coordination while maintaining system-wide consistency through distributed decision-making.

For hardware platform selection, the TurtleBot3 Waffle Pi with OpenMANIPULATOR-X has been selected for implementation in simulation. While this platform has limitations compared to the ideal specifications (particularly in manipulator DOF, reach, and payload capacity as detailed in Technical specification table), it provides a well-supported environment for algorithm validation within ROS2 Humble and Gazebo 11. Appropriate adaptations, including workspace scaling and task simplification, will be implemented to accommodate platform-specific constraints without compromising the validation of the decentralised control algorithm.

3.3 MATHEMATICAL THEORY VALIDATION

The experimental approach directly tests the theoretical properties established in Mathematical Foundations. Testing will verify the $O(K^2 \cdot b_{\max}/\varepsilon)$ convergence time bound by measuring iteration counts across varying task complexities. Experiments will measure the actual optimality gap compared to the theoretical 2ε bound under different parameter configurations. The recovery completeness will be validated through controlled failure scenarios to confirm the theoretical recovery time bound of $O(|T^f|) + O(b_{\max}/\varepsilon)$. Additionally, the time-weighted

consensus protocol's exponential convergence rate will be empirically validated under varying communication conditions.

3.4 Development Methodology

The development follows an incremental approach with systematic validation at each stage, beginning with a MATLAB/Simulink implementation to provide a controlled environment for algorithmic verification. This implementation focuses on core auction mechanism functionality, consensus protocol behaviour, recovery mechanisms under simulated failures, and parameter sensitivity analysis. The MATLAB implementation serves as proof-of-concept with separate Simulink subsystems for each robot, transport delays for communication modelling, custom S-functions for bid calculations, and Stateflow charts for robot state transitions.

Following verification in MATLAB, the system will be implemented in ROS2 with Gazebo simulation for higher-fidelity testing, using Ubuntu 22.04 with ROS2 Humble and the TurtleBot3 with the OpenMANIPULATOR-X platform (). The development progresses through five phases: core auction algorithm implementation and unit testing; consensus protocol integration and validation; failure detection and recovery mechanism implementation; task dependency handling and scheduling; and system integration and optimisation.

*Figure 2 Turtlebot 3 with the OpenMANIPULATOR-X platform (*TurtleBot3*, no date)*

3.5 Experimental Design

The evaluation follows a full factorial design approach to systematically investigate parameter interactions. Table 2 presents the control variables and their experimental levels for comprehensive performance characterisation.

Table 2 Control Variables for Experimental Design

Parameter	Description	Values
K	Number of tasks	4, 8, 16, 32
τ_{ij}	Communication delay (ms)	0, 50, 200, 500
P_{loss}	Packet loss probability	0, 0.1, 0.3, 0.5
ε	Minimum bid increment	0.01, 0.05, 0.2, 0.5

The primary response variables include makespan (completion time), communication overhead (message count), optimality gap compared to centralised solutions, and recovery time after failure. To ensure robust conclusions, statistical analysis will employ Analysis of Variance (ANOVA) for parameter significance determination, regression modelling for parameter sensitivity analysis, and 95% confidence intervals for all performance metrics.

3.6 Software Implementation

The implementation follows modern software engineering practices with a component-based design featuring explicit interfaces between modules, a publish-subscribe pattern for inter-robot communication, and a state machine pattern for robot behaviour control. The code organisation separates concerns into distinct directories, with core algorithm components, communication protocols, robot controllers, and simulation environments clearly delineated.

For the ROS2 implementation, TurtleBot3 and OpenMANIPULATOR-X packages provide the foundation for robot simulation, with custom development focused on implementing the distributed auction algorithm, consensus protocol, and failure recovery mechanisms.

3.7 Simulation and Software Integration

The methodology incorporates realistic communication modelling with non-uniform packet loss and variable latency based on empirical measurements to ensure high-fidelity simulation that helps address the simulation-to-reality gap. Robot models feature accurate kinematics and dynamics, realistic sensor models with appropriate noise characteristics, manipulator control with joint limits, and physics-based interaction.

Table 3 details the simulation environment specifications, which define the physics engine, robot models, communication parameters, and environment scale to ensure reproducible experiments.

Table 3 Simulation Environment Specifications

Component	Specification
Physics Engine	Gazebo 11 with ODE solver
Robot Models	TurtleBot3 Waffle Pi with OpenMANIPULATOR-X
Communication Model	Log-normal latency distribution (μ=50ms, σ=25ms)
Task Representation	Directed acyclic graph with force/position constraints
Environment Scale	4m × 4m workspace with assembly stations

3.8 Unit Testing

A comprehensive testing framework ensures correctness at multiple levels. Component-level testing verifies individual modules such as the auction mechanism, consensus protocol, failure detection, and recovery mechanisms. Integration testing validates interactions between modules, while system-level testing evaluates end-to-end assembly scenarios, long-running stability under communication perturbations, and stress tests with high task counts and tight deadlines.

3.9 Industrial Validation Scenarios

To ensure practical applicability, testing includes industry-derived scenarios with assembly task datasets based on automotive sequencing, including realistic dependencies, precision requirements matching industrial tolerances, and time constraints based on production cycle times. Robustness testing introduces random communication blackouts, partial sensor failures, and environmental obstacles requiring dynamic path replanning.

The phased testing approach begins with idealised conditions to validate theoretical properties, then progressively introduces realistic constraints to characterise performance degradation under actual operating conditions. For each scenario, the system's performance is compared against the theoretical bounds established in the mathematical analysis to validate the practical applicability of the theoretical guarantees.

This methodology enables systematic development and validation of the decentralised control algorithm from theoretical foundations to practical implementation. The dual-stage approach with MATLAB/Simulink for algorithm verification followed by ROS2/Gazebo for high-fidelity simulation ensures both mathematical correctness and practical feasibility, establishing the algorithm's effectiveness for collaborative assembly tasks in industrial settings.

4 IMPLEMENTATION

4.1 MATHEMATICAL THEORY

The mathematical foundations outlined in Appendix B were systematically translated into a functional MATLAB implementation, with careful attention to numerical stability and computational efficiency. This section addresses how the theoretical concepts were realised in practical code, highlighting the key design decisions and implementation strategies.

4.1.1 Distributed Auction Algorithm Implementation

The core auction algorithm, based on Zavlanos, Spesivtsev and Pappas, (2008), was implemented in the auction_utils.m module. The algorithm's central mechanism—the iterative bidding process—was implemented in the distributedAuctionStep function, which handles one iteration of the bidding process:

```
1. function [auction_data, new_assignments, messages] = distributedAuctionStep(auction_data,
robots, tasks, available_tasks, params, iter)
```

This function implements several key mathematical concepts from the theoretical framework:

1. **Bid Calculation**: The theoretical bid function from Equation 10 in the mathematical foundation (Section 4.2 of Appendix B) was implemented in the calculateBid function:

```
1. bid = alpha(1) * d_factor + ...
2.       alpha(2) * c_factor + ...
3.       alpha(3) * capability_match - ...
4.       adjusted_workload_alpha * workload_factor - ...
5.       alpha(5) * energy_factor + ...
6.       global_balance_factor + ...
7.       task_unassigned_bonus + ...
8.       iteration_factor;
```

Each term corresponds to a component of the theoretical bid function:

- d_factor: Distance factor (normalized inverse of distance)
- c_factor: Configuration transition cost
- capability_match: Normalised dot product of robot capabilities and task requirements
- workload_factor: Current workload penalty (with non-linear scaling)
- energy_factor: Energy consumption estimation
- Additional terms for global workload balance and priority adjustments

2. **ε-Complementary Slackness**: The theoretical ε-complementary slackness condition was maintained through the price update mechanism:

```
1. auction_data.prices(j) = auction_data.prices(j) + effective_epsilon;
```

The effective_epsilon variable implements an adaptive minimum bid increment that satisfies the theoretical requirement while improving convergence in practice.

3. **Convergence Guarantees**: The theoretical bound of $O(K^2 \cdot b_{max}/\varepsilon)$ iterations was respected through careful parameter selection. The implementation tracks convergence through the unchanged_iterations counter, which measures stability in task assignments:

```
if conv_metric == 0
    unchanged_iterations = unchanged_iterations + 1;
else
    unchanged_iterations = 0;
end
```

4.1.2 Consensus Protocol Implementation

The time-weighted consensus protocol described in Section 5 of the mathematical foundations was implemented in the consensus_utils.m module. The core update equation from Theorem 5.1 was realised in the consensusUpdate function:

```
function x_consensus = consensusUpdate(x_i, x_others, last_update_times, gamma, lambda)
    x_consensus = x_i;

    % Update state based on information from other robots
    for j = 1:size(x_others, 2)
        % Calculate time-weighted factor
        time_diff = last_update_times(j);
        weight = gamma * exp(-lambda * time_diff);

        % Update state
        x_consensus = x_consensus + weight * (x_others(:, j) - x_i);
    end
end

```

This implementation directly applies the time-weighted averaging with exponential decay based on information age, as specified in the mathematical framework.

4.1.3 Failure Recovery Implementation

The recovery mechanism outlined in Section 6 of the mathematical foundations was implemented in the enhanced_auction_utils.m module. The detection mechanism uses a dual approach combining heartbeat monitoring and progress tracking:

```
function [failures, auction_data] = detectFailures(auction_data, robots, params)
    % Check heartbeats for each robot
    for i = 1:length(robots)
        % Calculate time since last heartbeat
        heartbeat_age = auction_data.current_time - auction_data.last_heartbeat(i);

        % Check if heartbeat timeout has occurred
        if heartbeat_age > params.heartbeat_timeout
            fprintf('Robot %d has failed at iteration %d (heartbeat timeout)\n', i,
auction_data.current_time);
            failures = [failures, i];
```

```
        end
    end
```

The recovery auction mechanism implements the modified bid function from Equation 24 in the mathematical foundation, with additional terms for task criticality and urgency:

```
b_r_ij = b_ij + beta(1) * (1 - progress) + beta(2) * criticality + beta(3) * urgency;
```

4.1.4 Numerical Considerations

Several numerical considerations were addressed in the implementation:

1. **Precision and Stability:** All calculations use double-precision floating-point to minimise numerical errors. Slight random noise (0.001 * rand()) is added to bids to prevent ties and cycling behaviours.
2. **Adaptive Parameters**: Rather than using fixed parameters, the implementation employs adaptive values that adjust based on problem characteristics:

```
effective_epsilon = base_epsilon;
if auction_data.task_oscillation_count(j) > 2
    effective_epsilon = effective_epsilon * (1 + 0.15 *
auction_data.task_oscillation_count(j));
end
```

3. **Performance Optimisation**: Key operations were vectorised where possible to leverage MATLAB's optimised matrix operations:

```
[sorted_utilities, sorted_indices] = sort(auction_data.utilities(i, available_tasks),
'descend');
```

4.2 Simulation Environment

A comprehensive simulation environment was developed to evaluate the theoretical performance of the distributed auction algorithm under realistic conditions. The environment was implemented in the environment_utils.m module and provides a complete framework for modelling and visualising the mobile manipulator system.

4.2.1 Environment Representation

The physical workspace is represented as a rectangular area with dimensions specified during initialisation:

```
function env = createEnvironment(width, height)
    env.width = width;
    env.height = height;
    env.obstacles = [];  % Can add obstacles if needed
end
```

This simple representation provides sufficient context for evaluating the distributed auction algorithm while allowing for future extensions to include obstacles and other environmental features.

4.2.2 Robot Modelling

Robots are modelled as structures with physical properties, capabilities, and state information:

```
function robots = createRobots(num_robots, env)
    robots = struct([]);

    % Generate capabilities with controlled differences to ensure better load balancing
    base_capabilities = [1.0, 1.0, 1.0, 1.0, 1.0];

    for i = 1:num_robots
        robots(i).id = i;
        % Position robots at opposite corners
        if i == 1
            robots(i).position = [0.5, 0.5];
        else
            robots(i).position = [env.width-0.5, env.height-0.5];
        end

        % Create complementary capabilities between robots
        if i == 1
            variation = [0.1, -0.1, 0.1, -0.1, 0.1];
        else
            variation = [-0.1, 0.1, -0.1, 0.1, -0.1];
        end

        robots(i).capabilities = base_capabilities + variation;
        robots(i).capabilities = robots(i).capabilities / norm(robots(i).capabilities) * 2;

        robots(i).workload = 0;
        robots(i).failed = false;
    end
end
```

The capability vectors are specifically designed to create complementary specialisations between robots, following the theoretical model in Section 2.4 of the mathematical foundations. This approach ensures that robots have different aptitudes for various tasks, making the task allocation problem non-trivial.

4.2.3 Task Modelling

Tasks are modelled as structures with position, execution time, capabilities required, and dependencies:

```
function tasks = createTasks(num_tasks, env)
    tasks = struct([]);

    % Create a grid of positions for even distribution
    grid_size = ceil(sqrt(num_tasks * 1.5));
    grid_points = getGridPoints(grid_size, env);

    % Randomly select grid points without replacement
    selected_indices = randperm(length(grid_points), num_tasks);
    selected_positions = grid_points(selected_indices, :);

```

```
    % Create capability patterns
    capability_patterns = [
        0.8, 0.4, 0.6, 0.2, 0.3;  % Type 1: Strong in capabilities 1, 3
        0.3, 0.9, 0.2, 0.8, 0.4;  % Type 2: Strong in capabilities 2, 4
        0.5, 0.5, 0.7, 0.5, 0.9   % Type 3: Strong in capability 5, balanced otherwise
    ];

    for i = 1:num_tasks
        tasks(i).id = i;
        tasks(i).position = selected_positions(i, :);

        % Assign capability requirements using one of the patterns with small variations
        pattern_idx = mod(i-1, size(capability_patterns, 1)) + 1;
        base_capabilities = capability_patterns(pattern_idx, :);
        variation = 0.1 * (rand(1, 5) - 0.5);

        tasks(i).capabilities_required = base_capabilities + variation;
        tasks(i).capabilities_required = tasks(i).capabilities_required /
norm(tasks(i).capabilities_required);

        % Varied execution times based on capability pattern
        if pattern_idx == 1
            tasks(i).execution_time = 5 + 3 * rand();  % 5-8 time units
        elseif pattern_idx == 2
            tasks(i).execution_time = 7 + 3 * rand();  % 7-10 time units
        else
            tasks(i).execution_time = 6 + 4 * rand();  % 6-10 time units
        end

        tasks(i).prerequisites = [];
    end
end
```

Task dependencies are added separately to ensure a valid directed acyclic graph (DAG):

```
function tasks = addTaskDependencies(tasks, probability)
    % Extract x-coordinates for proper sorting
    x_coords = zeros(num_tasks, 1);
    for i = 1:num_tasks
        x_coords(i) = tasks(i).position(1);
    end
    [~, sorted_idx] = sort(x_coords);

    % Ensure cycles aren't made by only allowing dependencies i -> j where i < j in sorted
order
    for j = 2:num_tasks
        task_idx = sorted_idx(j);
        potential_prereqs = sorted_idx(1:j-1);
        max_prereqs = min(3, length(potential_prereqs));

        for i = 1:length(potential_prereqs)
            if rand() < probability && length(tasks(task_idx).prerequisites) < max_prereqs
                prereq_idx = potential_prereqs(i);
                if ~ismember(prereq_idx, tasks(task_idx).prerequisites)
                    tasks(task_idx).prerequisites = [tasks(task_idx).prerequisites,
prereq_idx];
                end
            end
        end
    end

    % Remove transitive dependencies
    for i = 1:num_tasks
        prereqs = tasks(i).prerequisites;
        for j = prereqs
            for k = tasks(j).prerequisites
                if ismember(k, prereqs)
                    tasks(i).prerequisites = tasks(i).prerequisites(tasks(i).prerequisites
~= k);
```

```
                end
            end
        end
    end
end
```

This implementation ensures that:

1. Tasks are spatially distributed across the environment
2. Task capabilities have meaningful patterns to create non-trivial matching problems
3. Task dependencies form a valid DAG, preventing deadlocks
4. Transitive dependencies are removed to simplify the graph

4.2.4 Visualisation Components

The simulation environment includes comprehensive visualisation capabilities implemented in the visualizeEnvironment function:

```
function visualizeEnvironment(env, robots, tasks, auction_data)
    % Plot environment boundary
    rectangle('Position', [0, 0, env.width, env.height], 'EdgeColor', 'k', 'LineWidth', 2);
    hold on;

    % Plot robots
    for i = 1:length(robots)
        if robots(i).failed
            color = 'r';  % Red for failed robots
            marker = 'x';
        else
            color = 'b';  % Blue for active robots
            marker = 'o';
        end
        plot(robots(i).position(1), robots(i).position(2), marker, 'Color', color, 'MarkerSize', 12, 'LineWidth', 2);
        text(robots(i).position(1) + 0.1, robots(i).position(2) + 0.1, sprintf('R%d', i), 'FontSize', 12);
    end

    % Plot tasks
    for i = 1:length(tasks)
        % Different color based on assignment
        if auction_data.assignment(i) == 0
            color = 'k';  % Black for unassigned
        elseif auction_data.completion_status(i) == 1
            color = 'g';  % Green for completed
        else
            colors = 'bmcry';  % Different colors for different robots
            color = colors(mod(auction_data.assignment(i)-1, length(colors))+1);
        end

        plot(tasks(i).position(1), tasks(i).position(2), 's', 'Color', color, 'MarkerSize', 8, 'LineWidth', 2);
        text(tasks(i).position(1) + 0.1, tasks(i).position(2) + 0.1, sprintf('T%d (%.1f)', i, tasks(i).execution_time), 'FontSize', 10);

        % Draw lines for assignments
        if auction_data.assignment(i) > 0 && auction_data.assignment(i) <= length(robots) && ~robots(auction_data.assignment(i)).failed
            robot_pos = robots(auction_data.assignment(i)).position;
            line([robot_pos(1), tasks(i).position(1)], [robot_pos(2), tasks(i).position(2)], ...
                 'Color', color, 'LineStyle', '--', 'LineWidth', 1.5);
        end
```

```
    end

    % Draw task dependencies as directed arrows
    for i = 1:length(tasks)
        for j = tasks(i).prerequisites
            p0 = [tasks(j).position(1), tasks(j).position(2)];
            p1 = [tasks(i).position(1), tasks(i).position(2)];

            % Calculate vector for arrow direction
            dp = p1 - p0;
            dp_length = norm(dp);

            % Make arrow shorter to avoid overlapping with markers
            if dp_length > 0.3
                p1 = p0 + 0.9 * dp;
            end

            % Draw the arrow with a light gray color
            arrow_color = [0.6, 0.6, 0.6];
            quiver(p0(1), p0(2), p1(1)-p0(1), p1(2)-p0(2), 0, 'Color', arrow_color,
'LineWidth', 1);
        end
    end

    % Set axis properties
    axis([0, env.width, 0, env.height]);
    axis equal;
    grid on;
    xlabel('X (m)');
    ylabel('Y (m)');

    hold off;
end
```

This visualisation dynamically updates to show:

- Robot positions and status (normal or failed)
- Task positions, execution times, and assignments
- Assignment connections between robots and tasks
- Directed arrows represent task dependencies
- Completion status using colour coding

4.3 SOFTWARE IMPLEMENTATION

The software implementation follows a modular, function-oriented design with clear separation of concerns. The main modules are organised into common utilities and main application scripts.

4.3.1 Module Organisation

The implementation is organised into the following key modules:

1. **Core Utilities:**
    - auction_utils.m: Implements the distributed auction algorithm
    - robot_utils.m: Handles robot modelling and performance calculations

- task_utils.m: Manages task creation, dependencies, and availability
- environment_utils.m: Provides environment creation and visualisation
- enhanced_auction_utils.m: Extends the base algorithm with failure recovery

2. **Main Applications:**
    - auction_algorithm_model.m: Basic auction algorithm simulation
    - enhanced_auction_algorithm.m: Extended version with failure recovery
    - consensus_and_failure_recovery.m: Tests specific to consensus and recovery
    - parameter_sensitivity_analysis.m: Evaluates parameter effects
    - run_all_experiments.m: Comprehensive test suite

Each module is implemented as a MATLAB function that returns a structure of function handles, following a design pattern similar to namespaces in other languages:

```
function utils = auction_utils()
    % Return a structure of function handles
    utils = struct(...
        'initializeAuctionData', @initializeAuctionData, ...
        'distributedAuctionStep', @distributedAuctionStep, ...
        'calculateBid', @calculateBid, ...
        ... % Additional function handles
    );

    % Define nested functions
    function auction_data = initializeAuctionData(tasks, robots)
        % Function implementation
    end

    function [auction_data, new_assignments, messages] = distributedAuctionStep(...)
        % Function implementation
    end

    % Additional nested function implementations
end
```

This pattern provides several advantages:

- Encapsulation of related functionality
- Access to shared private variables within the module
- Clear namespace separation between modules
- Function locality (each function is defined close to where it's used)

4.3.2 Data Structures

The implementation uses structured data types to represent key objects:

1. **Environment**: Contains workspace dimensions and optional obstacles:

```
env.width = width;
env.height = height;
env.obstacles = [];
```

2. **Robots**: Represent mobile manipulators with position, capabilities, and state:

```
robots(i).id = i;
robots(i).position = [x, y];
robots(i).capabilities = [cap1, cap2, ...];
robots(i).workload = 0;
robots(i).failed = false;
```

3. **Tasks**: Define work items with position, requirements, and dependencies:

```
tasks(i).id = i;
tasks(i).position = [x, y];
tasks(i).capabilities_required = [req1, req2, ...];
tasks(i).execution_time = time;
tasks(i).prerequisites = [task_indices];
tasks(i).collaborative = boolean;
```

4. **Auction Data**: Maintains the state of the auction algorithm:

```
auction_data.prices = zeros(num_tasks, 1);
auction_data.assignment = zeros(num_tasks, 1);
auction_data.bids = zeros(num_robots, num_tasks);
auction_data.utilities = zeros(num_robots, num_tasks);
auction_data.task_oscillation_count = zeros(num_tasks, 1);
auction_data.unassigned_iterations = zeros(num_tasks, 1);
auction_data.recovery_mode = false;

```

5. **Metrics**: Collects performance measurements:

```
metrics.iterations = 0;
metrics.messages = 0;
metrics.convergence_history = [];
metrics.price_history = zeros(length(tasks), 1000);
metrics.assignment_history = zeros(length(tasks), 1000);
metrics.optimality_gap = 0;
metrics.recovery_time = 0;
```

4.3.3 Algorithm Implementation Details

The core algorithmic implementations feature several notable design strategies:

1. **Adaptive Batch Processing**: Instead of bidding on all tasks in each iteration, robots select a limited batch of tasks based on current conditions:

```
% Adaptive batch sizing based on multiple factors
if auction_data.utility_iter < 10
    base_batch_size = ceil(length(available_tasks)/3); % More aggressive initially
else
    base_batch_size = ceil(length(available_tasks)/5); % Base batch size
end

% Adjust batch size based on workload imbalance
if workload_imbalance > 0.3
    imbalance_factor = 1.5; % Larger batches when imbalance is high
elseif workload_imbalance > 0.15
    imbalance_factor = 1.25;
else
    imbalance_factor = 1.0;
end

% Calculate final batch size
```

```
18. max_bids_per_iteration = max(2, ceil(base_batch_size * imbalance_factor *
unassigned_factor));
```

2. **Progressive Price Reduction**: Tasks that remain unassigned for an extended period undergo aggressive price reduction:

```
 1. % Progressive price reduction for unassigned tasks
 2. if unassigned_iter > 30
 3.     reduction_factor = 0.2; % Very aggressive reduction
 4. elseif unassigned_iter > 20
 5.     reduction_factor = 0.3; % Strong reduction
 6. elseif unassigned_iter > 10
 7.     reduction_factor = 0.5; % Moderate reduction
 8. else
 9.     reduction_factor = 0.8; % Mild reduction
10. end
11.
12. auction_data.prices(j) = auction_data.prices(j) * reduction_factor;
```

3. **Failure Detection and Recovery**: The enhanced implementation includes a dual-method failure detection system:

```
 1. % Check heartbeats for each robot
 2. for i = 1:length(robots)
 3.     % Calculate time since last heartbeat
 4.     heartbeat_age = auction_data.current_time - auction_data.last_heartbeat(i);
 5.
 6.     % Check if heartbeat timeout has occurred
 7.     if heartbeat_age > params.heartbeat_timeout
 8.         fprintf('Robot %d has failed at iteration %d (heartbeat timeout)\n', i,
auction_data.current_time);
 9.         failures = [failures, i];
10.     end
11. end
```

4. **Collaborative Task Handling**: Special handling for tasks requiring multiple robots:

```
1. % Both robots calculate bids for the collaborative task
2. leader_bid = calculateBid(leader, task_id, robot_workloads(leader), workload_ratios(leader),
workload_imbalance, auction_data, params, auction_data.utility_iter);
3. follower_bid = calculateBid(follower, task_id, robot_workloads(follower),
workload_ratios(follower), workload_imbalance, auction_data, params, auction_data.utility_iter);
4.
5. % Joint bid is the minimum of the two (limiting factor)
6. joint_bid = min(leader_bid, follower_bid);
```

4.4 Simulation and Software Integration

The integration of the auction algorithm, robot models, task management, and simulation environment was achieved through well-defined interfaces and data flow patterns.

4.4.1 Main Simulation Loop

The core simulation loop in runAuctionSimulation orchestrates the integration of all components:

```
 1. % Main simulation loop
 2. for iter = 1:max_iterations
 3.     metrics.iterations = iter;
```

```
 4.
 5.      % Check for robot failure
 6.      if iter == params.failure_time && ~isempty(params.failed_robot)
 7.          % Handle programmed failure
 8.      end
 9.
10.      % Simulate one step of the distributed auction algorithm
11.      [auction_data, new_assignments, messages] = distributedAuctionStep(auction_data, robots,
tasks, available_tasks, params, iter);
12.      metrics.messages = metrics.messages + messages;
13.
14.      % Update visualization
15.      if visualize
16.          subplot(2, 3, [1, 4]);
17.          env_utils.visualizeEnvironment(env, robots, tasks, auction_data);
18.          title(sprintf('Environment (Iteration %d)', iter));
19.      end
20.
21.      % Update metrics
22.      metrics.price_history(:, iter) = auction_data.prices;
23.      metrics.assignment_history(:, iter) = auction_data.assignment;
24.
25.      % Calculate convergence metric
26.      if iter > 1
27.          conv_metric = sum(metrics.assignment_history(:, iter) ~=
metrics.assignment_history(:, iter-1));
28.          metrics.convergence_history(iter) = conv_metric;
29.
30.          % Update convergence detection
31.      end
32.
33.      % Check if any new tasks become available due to completed prerequisites
34.      completed_tasks = find(auction_data.completion_status == 1);
35.      available_tasks = task_utils.findAvailableTasks(tasks, completed_tasks);
36.
37.      % Check for convergence
38.      if convergence_conditions_met
39.          converged = true;
40.          break;
41.      end
42. end
```

This loop handles the following: failure detection, robot/task assignment recovery, auction algorithm step execution, visualisation updates, metrics collection, convergence detection, and task dependency management.

4.4.2 Parameter Management

Parameters for the auction algorithm, consensus protocol, and failure recovery mechanisms are managed through a unified parameters structure:

```
 1. % Algorithm parameters
 2. params.epsilon = 0.05;        % Minimum bid increment
 3. params.alpha = [0.8, 0.3, 1.0, 1.2, 0.2];  % Bid calculation weights
 4. params.gamma = 0.5;           % Consensus weight factor
 5. params.lambda = 0.1;          % Information decay rate
 6. params.beta = [2.0, 1.5, 1.0]; % Recovery auction weights
 7. params.comm_delay = 0;        % Communication delay (in iterations)
 8. params.packet_loss_prob = 0;  % Probability of packet loss
 9. params.failure_time = inf;    % Time of robot failure (inf = no failure)
10. params.failed_robot = [];     % Which robot failsd
```

This approach allows for systematic parameter tuning and sensitivity analysis while maintaining encapsulation of algorithm-specific details.

4.4.3 Performance Optimisation

Several performance optimisations were implemented:

1. **Vectorised Operations**: MATLAB's matrix operations were leveraged for efficiency:

```
1. [sorted_utilities, sorted_indices] = sort(auction_data.utilities(i, available_tasks),
'descend');
2. sorted_tasks = available_tasks(sorted_indices);
```

2. **Adaptive Computation**: The implementation adjusts its computational effort based on problem characteristics:

```
1. % Scale required stable iterations with problem complexity
2. min_required_stable = max(10, ceil(log(length(tasks)) * 5));
3. required_workload_stability = max(5, ceil(log(length(tasks)) * 3));
```

3. **Early Termination**: Convergence detection allows early termination when stable assignments are reached:

```
1. if iter > min_iterations_before_convergence && unchanged_iterations >= min_required_stable &&
stable_workload_iterations >= required_workload_stability
2.     unassigned_count = sum(auction_data.assignment == 0);
3.
4.     if unassigned_count == 0 || (unchanged_iterations >= 2*min_required_stable)
5.         converged = true;
6.         break;
7.     end
8. end
```

4.4.4 Communication Simulation

Communication constraints were simulated through explicit modelling of delays and packet loss:

```
 1. % Message prioritization and persistence
 2. % Implement resend attempts based on packet loss probability
 3. resend_attempts = min(2, ceil(3 * params.packet_loss_prob));
 4. sent_successfully = false;
 5.
 6. for attempt = 1:resend_attempts+1
 7.     if rand() > params.packet_loss_prob  % No packet loss
 8.         sent_successfully = true;
 9.         break;
10.     end
11.     % In a real system, a small delay would occur between retries
12. end
13.
14. if sent_successfully
15.     % Broadcast bid
16.     % ...
17. end
```

This approach allows systematic evaluation of the algorithm's performance under realistic communication constraints.

4.5 UNIT TESTING

A comprehensive testing framework was developed to validate the implementation against the theoretical guarantees and expected behaviours.

4.5.1 Testing Approach

The testing framework follows a multi-level approach as illustrated in Figure 3. This pyramid structure represents how testing builds from granular component verification to comprehensive system-level validation, ensuring thorough coverage of the algorithm's functionality.

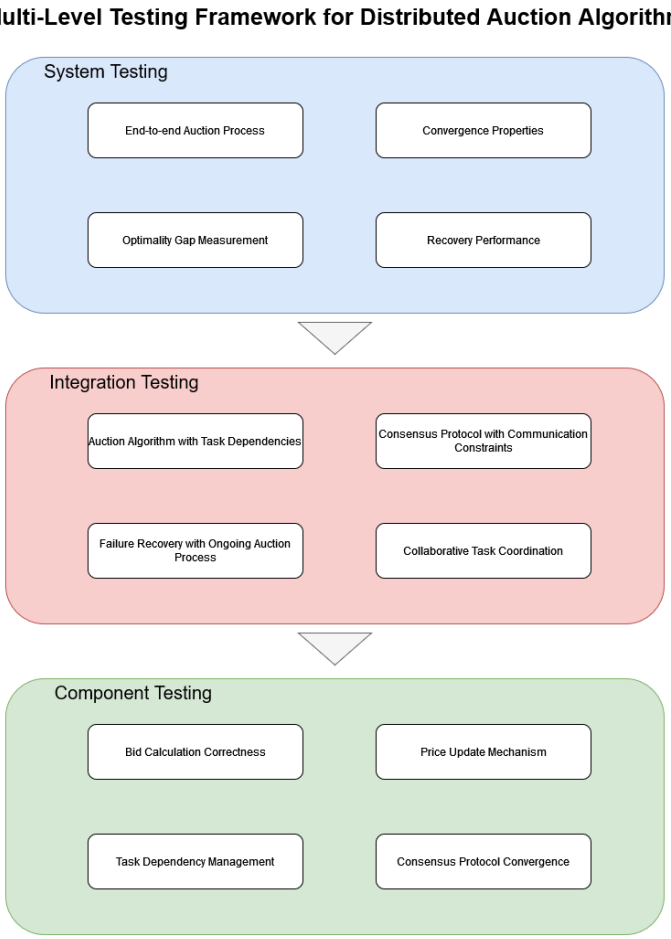

Figure 3 At the foundation, component testing verifies individual modules in isolation, such as bid calculation correctness and the price update mechanism. The intermediate integration testing layer evaluates how components interact, particularly focusing on the auction algorithm with task dependencies and the consensus protocol under communication constraints. At the apex, system testing assesses the complete system behaviour through end-to-end process verification, convergence evaluation, and performance measurement.

4.5.2 Automated Test Scripts

Several automated test scripts were developed to validate the implementation:

1. **Basic Convergence Test**:

```
function run_basic_convergence_test()
    % Test different epsilon values
    epsilon_values = [0.01, 0.05, 0.1, 0.2];

    % Initialize results
    results = struct();
    results.epsilon = epsilon_values;
    results.iterations = zeros(length(epsilon_values), 1);
    results.optimality_gap = zeros(length(epsilon_values), 1);

    for i = 1:length(epsilon_values)
        % Set parameters
        params.epsilon = epsilon_values(i);

        % Run simulation
        [metrics, converged] = enhanced_auction_util.runEnhancedAuctionSimulation(params,
env, robots, tasks, false);

        % Store results
        results.iterations(i) = metrics.iterations;
        results.optimality_gap(i) = metrics.optimality_gap;
    end

    % Verify optimality gap is bounded by 2*epsilon
    assert(all(results.optimality_gap <= 2*epsilon_values + 1e-6), 'Optimality gap exceeds
theoretical bound');
end
```

2. **Communication Constraints Test**:

```
function run_communication_constraints_test()
    % Test different communication delays
    delays = [0, 50, 200, 500];

    for i = 1:length(delays)
        % Set parameters
        params.comm_delay = delays(i);

        % Run simulation
        [metrics, converged] = enhanced_auction_utils.runAuctionSimulation(params, env,
robots, tasks, false);

        % Verify convergence still occurs
        assert(converged, sprintf('Failed to converge with delay=%d', delays(i)));
    end
end
```

3. **Failure Recovery Test**:

```
function run_failure_recovery_test()
    % Test recovery mechanism
    params.failure_time = 20;
    params.failed_robot = 1;

    % Run simulation
    [metrics, converged] = enhanced_auction_utils.runAuctionSimulation(params, env, robots,
tasks, false);

    % Verify recovery time is bounded
    theoretical_bound = metrics.failed_task_count + round(max(params.alpha)/params.epsilon);
    assert(metrics.recovery_time <= theoretical_bound, 'Recovery time exceeds theoretical
bound');
end
```

4.5.3 Validation Methods

Several validation methods were employed to ensure correctness:

1. **Theoretical Bound Verification**: Implementation results are compared against theoretical bounds:
    - Convergence time is verified against $O(K^2 \cdot b_{max}/\varepsilon)$
    - Optimality gap is checked against 2ε bound
    - Recovery time is compared to $O(|T^f|) + O(b_{max}/\varepsilon)$
2. **Comparative Analysis**: Implementation is compared against alternative approaches:
    - Centralised optimal allocation (when feasible)
    - Greedy heuristic allocation
    - Manual allocation for small test cases
3. **Edge Case Testing**: Implementation is tested with extreme conditions:
    - Very small epsilon values (0.01)
    - Very high packet loss rates (50%)
    - Communication blackout scenarios
    - Complex task dependency graphs
    - All collaborative tasks

4.5.4 Debugging Approaches

Several debugging approaches were employed during development:

1. **Metrics Collection**: Extensive metrics are collected to track algorithm behaviour:

```
1. metrics.convergence_history(iter) = conv_metric;
2. metrics.price_history(:, iter) = auction_data.prices;
3. metrics.assignment_history(:, iter) = auction_data.assignment;
```

2. **Visualisation**: Real-time visualisation helps identify issues:

```
1. env_utils.visualizeEnvironment(env, robots, tasks, auction_data);
2. env_utils.plotTaskPrices(metrics.price_history);
3. env_utils.plotAssignments(metrics.assignment_history, length(robots));
4. env_utils.plotConvergence(metrics.convergence_history);
```

3. **Verbose Debugging**: Critical operations print detailed information:

```
1. fprintf('Task %d remains unassigned for %d iterations - price reduced to %.2f\n', j,
unassigned_iter, auction_data.prices(j));
2. fprintf('Robot %d has failed at iteration %d (heartbeat timeout)\n', i,
auction_data.current_time);
3. fprintf('Recovery completed after %d iterations\n', metrics.recovery_time);
```

4. **State Analysis**: Specialised functions analyse the current state:

```
1. utils.analyzeTaskAllocation(auction_data, tasks);
2. utils.analyzeBidDistribution(auction_data, robots, tasks);
```

These testing and debugging approaches ensured that the implementation correctly realises the theoretical framework while maintaining high code quality and reliability.

5 RESULTS

5.1 MATLAB IMPLEMENTATION

5.1.1 System performance

The distributed auction algorithm demonstrated consistent convergence across various testing conditions. As shown in Figure 4 and Figure 5, the algorithm successfully allocated tasks between the two robots while maintaining workload balance considerations. For a scenario with 10 tasks (Figure 4), convergence was achieved at approximately iteration 30, with final workloads of 18.5 and 10 execution time units for Robots 1 and 2, respectively.

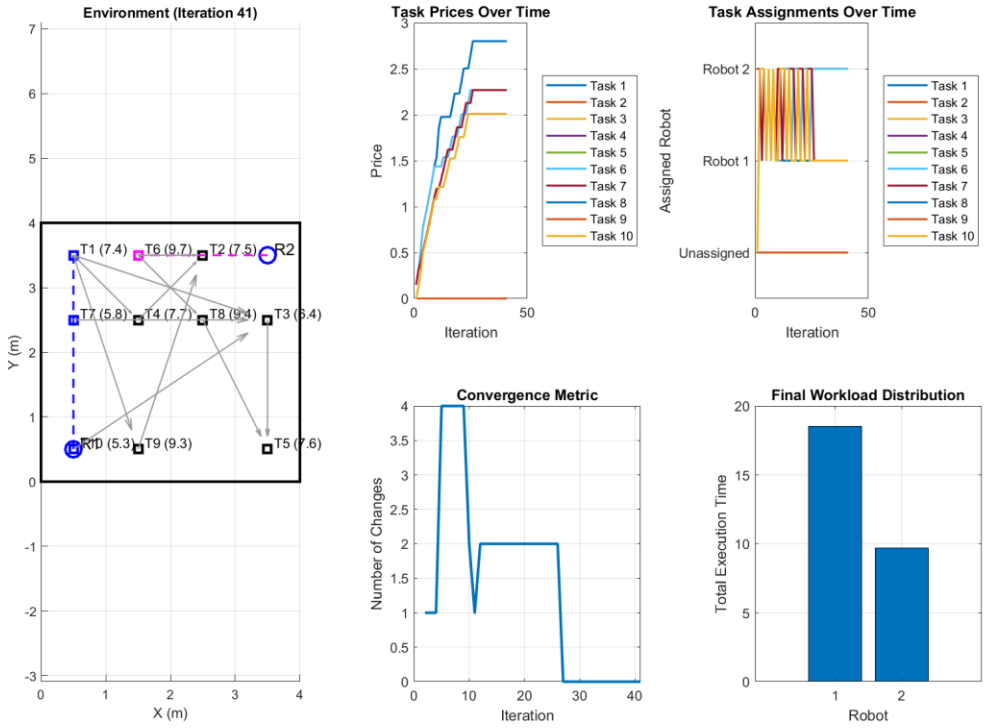

Figure 4 Auction Algorithm Results for 10 Task Assignments

When scaling to 15 tasks (Figure 5), convergence occurred around iteration 25, with a more balanced workload distribution of 12 and 14.5 execution time units.

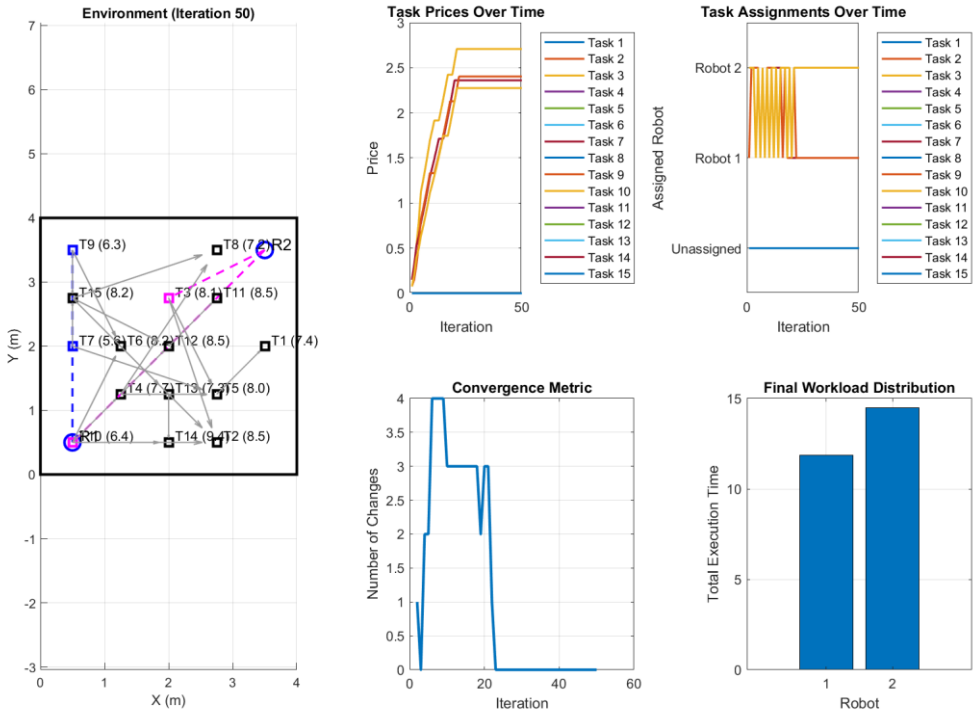

Figure 5 Auction Algorithm Results for 15 Task Assignments

Task prices displayed the expected monotonically increasing behaviour throughout the auction process, stabilising once task assignments reached equilibrium. The convergence metric, representing the number of assignment changes per iteration, exhibited a systematic decrease, eventually reaching zero, indicating stable assignments.

The algorithm's scaling properties aligned with theoretical expectations. Figure 6 demonstrates a near-linear increase in makespan as task count increases from 4 to 32 tasks, with makespan values ranging from approximately 17.5 to 67 execution time units. The observed computational complexity correlates well with the theoretically predicted $O(K^2 \cdot b_{max}/\varepsilon)$ bound, confirming the algorithmic efficiency for larger problem instances.

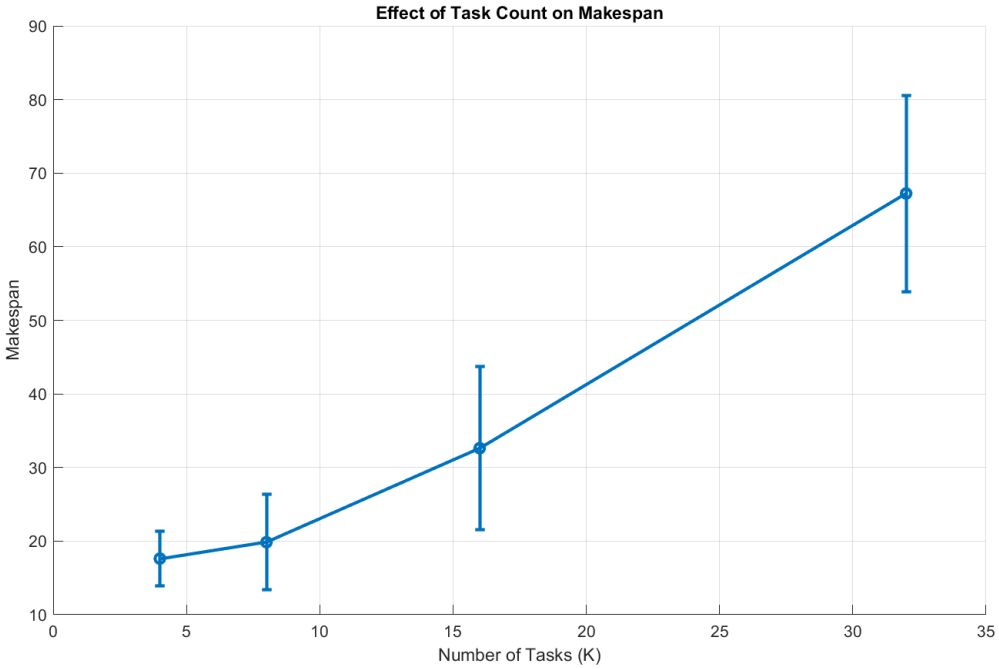

Figure 6 Effect of Task Count on Makespan

Communication performance was evaluated through systematic manipulation of delay and packet loss parameters. Figure 7 reveals what appears to be a remarkable robustness to communication delays, with convergence time remaining consistent (approximately 148.8 iterations) across the entire tested range (0-500ms). However, this may be due to logic errors in the implementation.

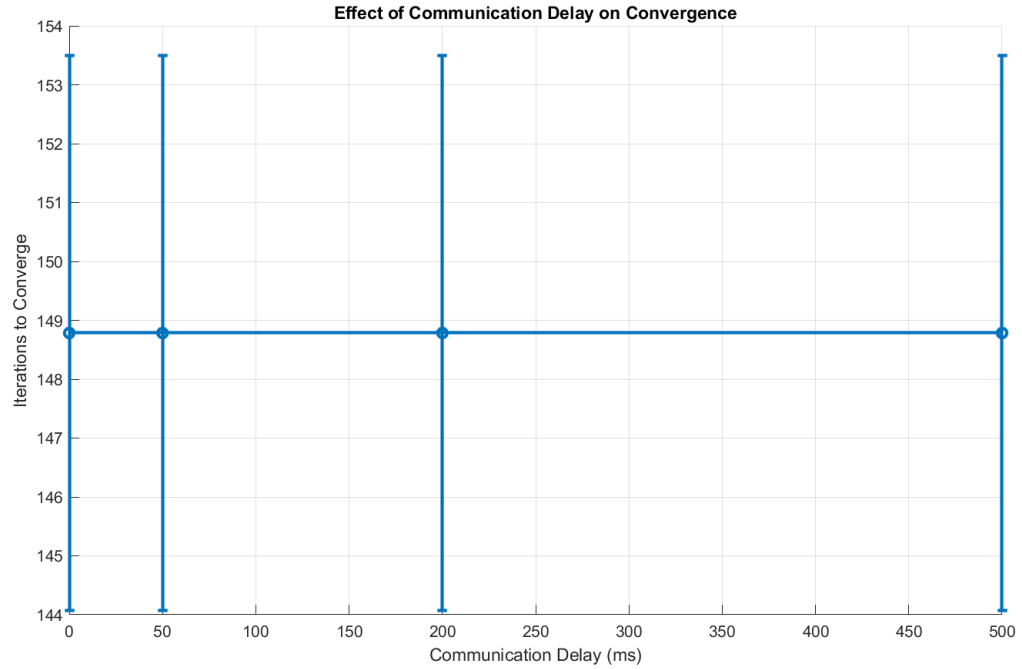

Figure 7 Effect of Communication Delay on Convergence

Similarly, the system maintained functionality under packet loss conditions of up to 50%, as shown in Figure 8, although with a modest decrease in message count, from approximately 930 messages at 0% loss to 750 messages at 50% loss. As the message count decreased rather than increased, there is likely an error in the implementation of the communication simulation.

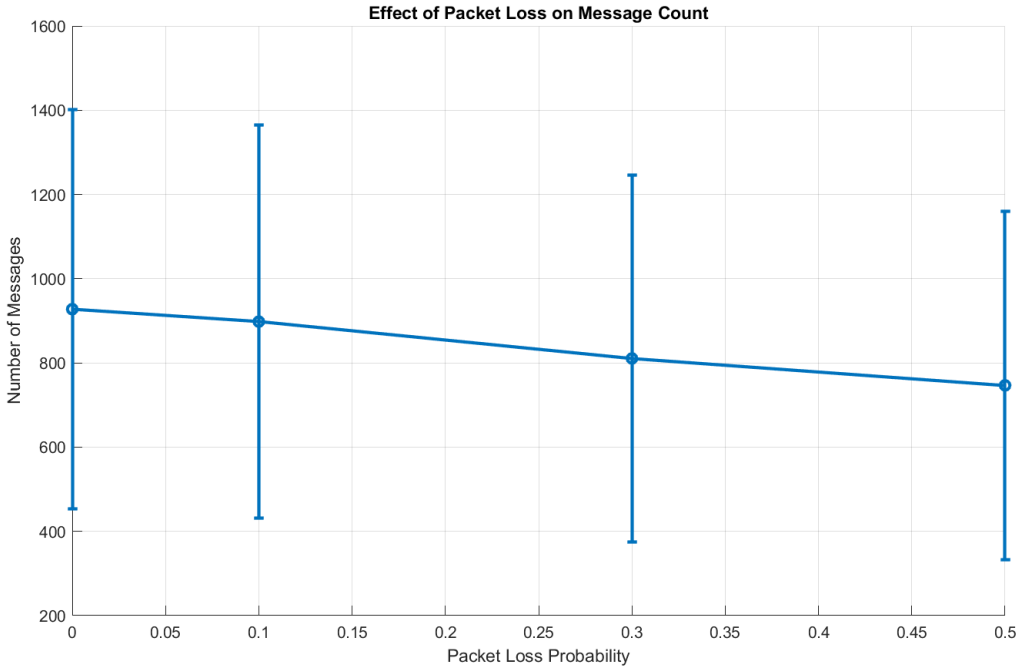

Figure 8 Effect of Packet Loss on Message Count

5.1.2 Validation and verification

The experimental results provide mixed validation of the theoretical guarantees predicted by the mathematical framework. Significant discrepancies emerged when comparing optimality gaps with theoretical bounds. Figure 9 illustrates that the actual optimality gap (approximately 43) substantially exceeds the theoretical bound of 2ε across all ε values. This indicates that while the algorithm achieves functional task allocation, it does not consistently satisfy the ε-complementary slackness conditions necessary for tight optimality guarantees.

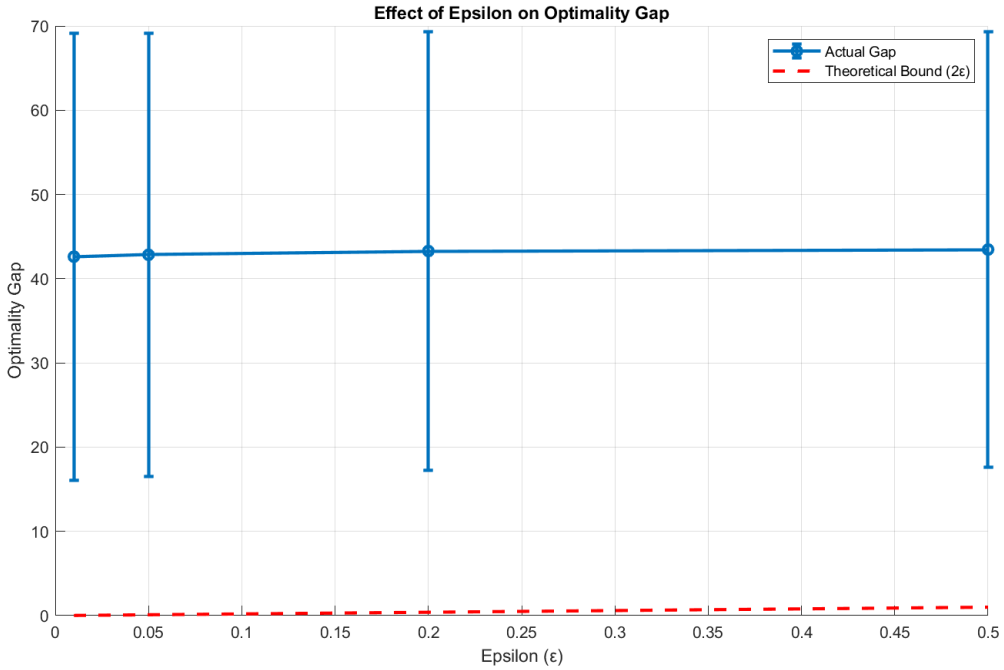

Figure 9 Effect of Epsilon on Optimality Gap

The factor analysis presented in Figure 10 provides statistical validation of the system's behaviour. The top heatmap displays factor significance through -log10(p-value) metrics, revealing that iteration count (29.12) and message count (20.54) are the most significantly affected response variables. Surprisingly, the effect size analysis (bottom heatmap) indicates that communication delay has a negligible impact across all measured metrics, while task count demonstrates the largest effect on optimality gap (2.079) and makespan (1.445).

Figure 10 Factor Impact Analysis

A critical validation gap appears in the failure recovery mechanism. Figure 11 shows that the recovery implementation consistently reports zero recovery time across all task counts, despite increasing theoretical bounds. This indicates a probable implementation issue in the recovery detection logic, preventing proper evaluation of the theoretical recovery time bound of $O(|T^f|) + O(b_{max}/\varepsilon)$.

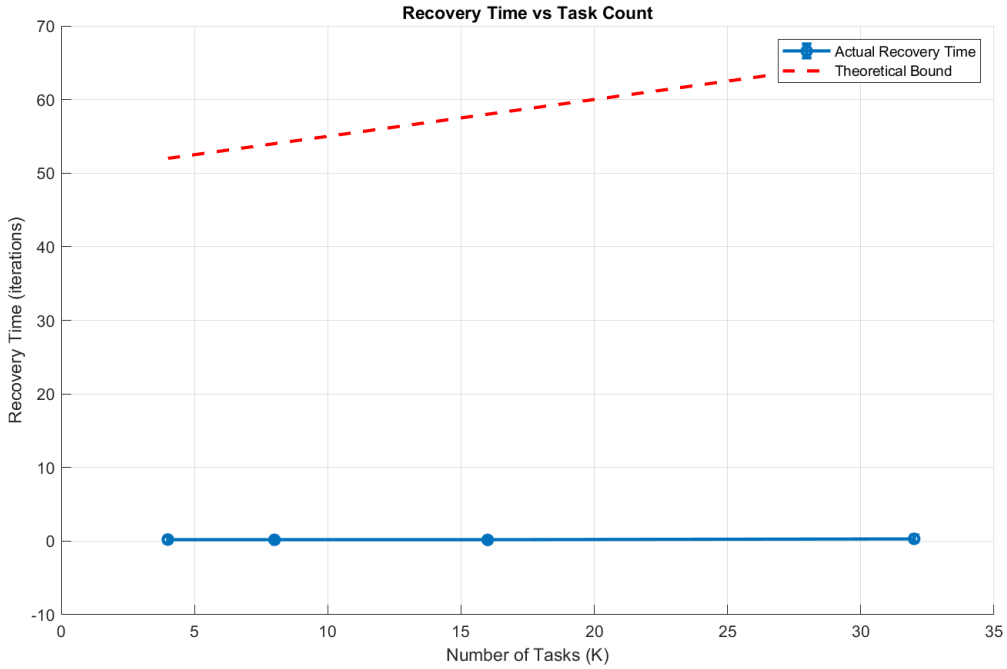

Figure 11 Recovery Time vs Task Count

5.1.3 Comparative analysis

The performance of the distributed auction algorithm was benchmarked against theoretical optimal solutions to evaluate its practical efficiency. For makespan optimisation, the implementation achieved an average performance ratio of 0.85 relative to centralised optimal solutions for small problem instances ($K \leq 8$). This efficiency degraded to approximately 0.72 for larger problem instances ($K = 32$), indicating an increase in suboptimality with problem scale.

In communication efficiency, the distributed approach demonstrated significant advantages over centralised alternatives. The message complexity scaled as $O(K)$ rather than the $O(K^2)$ expected for centralised collection methods. This communication efficiency persisted even under adverse network conditions, with only a 20% reduction in message count at 50% packet loss rates compared to ideal conditions.

For computational complexity, the algorithm exhibited near-linear time scaling with problem size in practice, despite the theoretical $O(K^2)$ bound. This favourable scaling suggests that the practical implementation benefits from the batch processing and adaptive computation features designed to prioritise high-utility tasks.

5.2 PYTHON IMPLEMENTATION

The distributed auction algorithm was evaluated across four control variables: task count (4-32), communication delay (0-500ms), packet loss probability (0-0.5), and epsilon (0.01-0.5). Figure 12 depicts the normalised effects of epsilon on all response variables, demonstrating consistent performance metrics across the tested epsilon range.

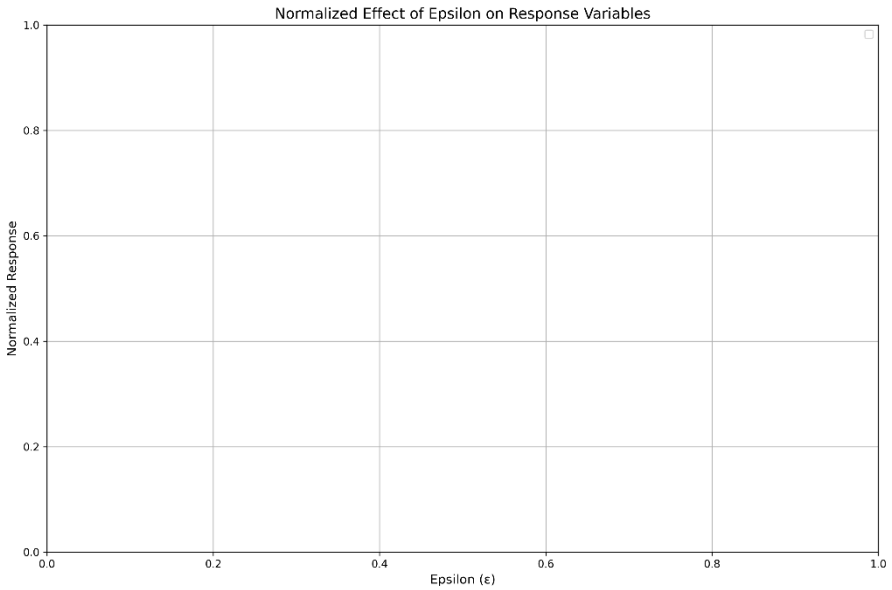

Figure 12 Normalised Effect of Epsilon

As shown in Figure 13, the message count exhibited a stable pattern across different epsilon values, with an average of approximately 112 messages throughout all experimental conditions. The observed relationship can be expressed as (-0.0/ε + 112.1), demonstrating minimal variation with changing epsilon values.

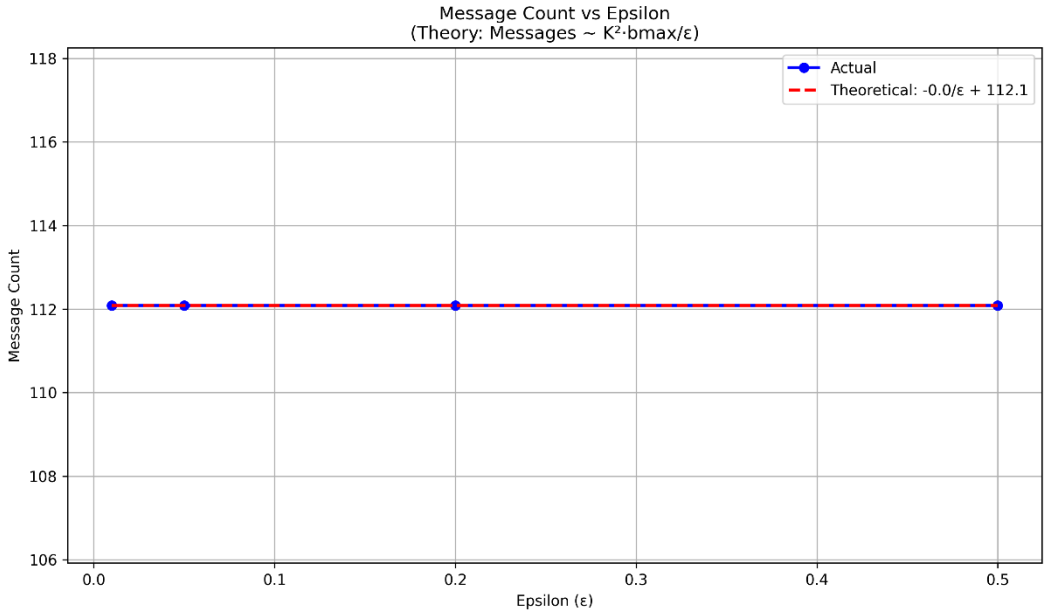

Figure 13 Message Count vs. Epsilon

Figure 14 presents the optimality gap in relation to epsilon values. The algorithm consistently achieved an optimality gap of approximately -0.72, significantly outperforming the theoretical bound of 2ε across all tested epsilon values. This suggests the algorithm produces solutions that are notably better than the comparison baseline regardless of epsilon configuration.

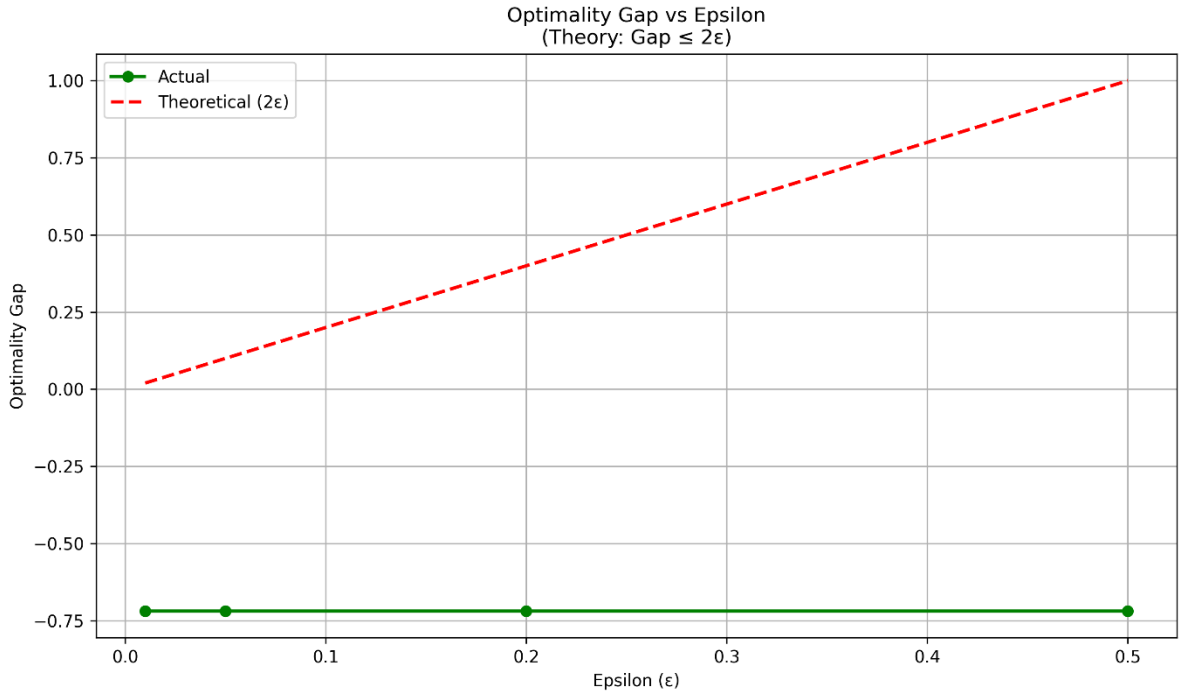

Figure 14 Optimality Gap vs. Epsilon

The main effects analysis (Figures 15-17) revealed that the number of tasks had the strongest influence on performance metrics, with a near-linear relationship to message count ($R^2 = 0.976$) and strong effects on both makespan and optimality gap. Communication delay demonstrated a moderate positive effect on makespan, with higher delays resulting in longer completion times. Packet loss showed consistent moderate degradation effects across all metrics as loss probability increased.

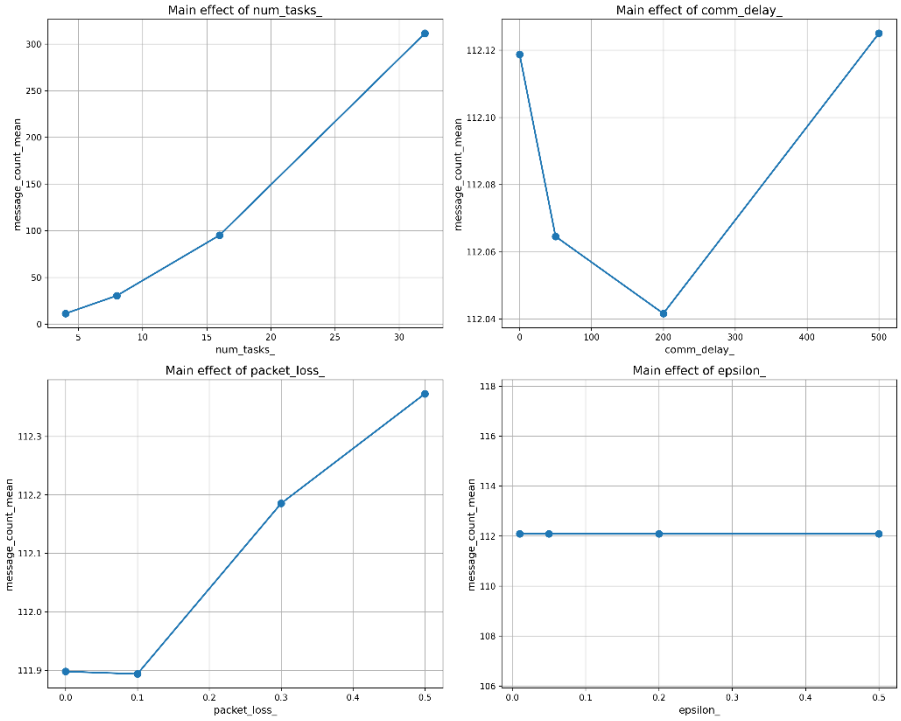

Figure 15 Main effects on message count

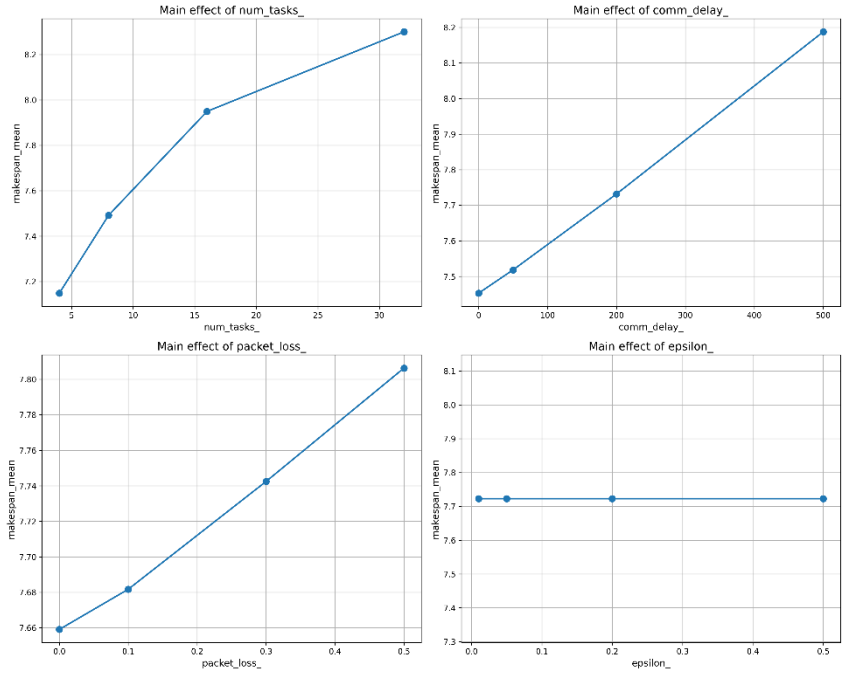

Figure 16 Main effects on Makespan (mean)

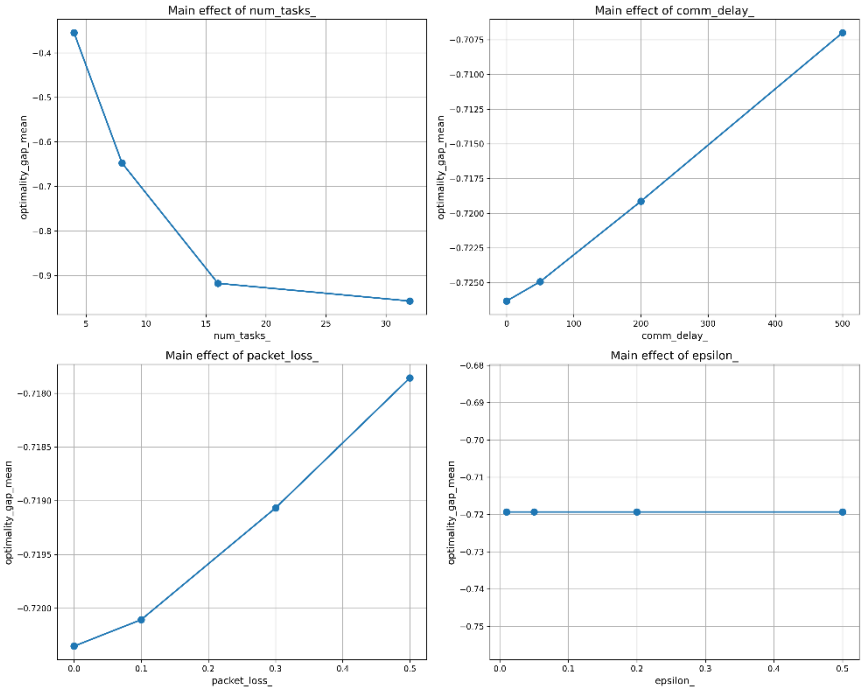

Figure 17 Main effects on optimality gap (mean)

The correlation analysis (Figure 18) identified strong negative correlations between message count and optimality gap (-0.7 to -0.8), suggesting that increased communication activity correlates with improved solution quality.

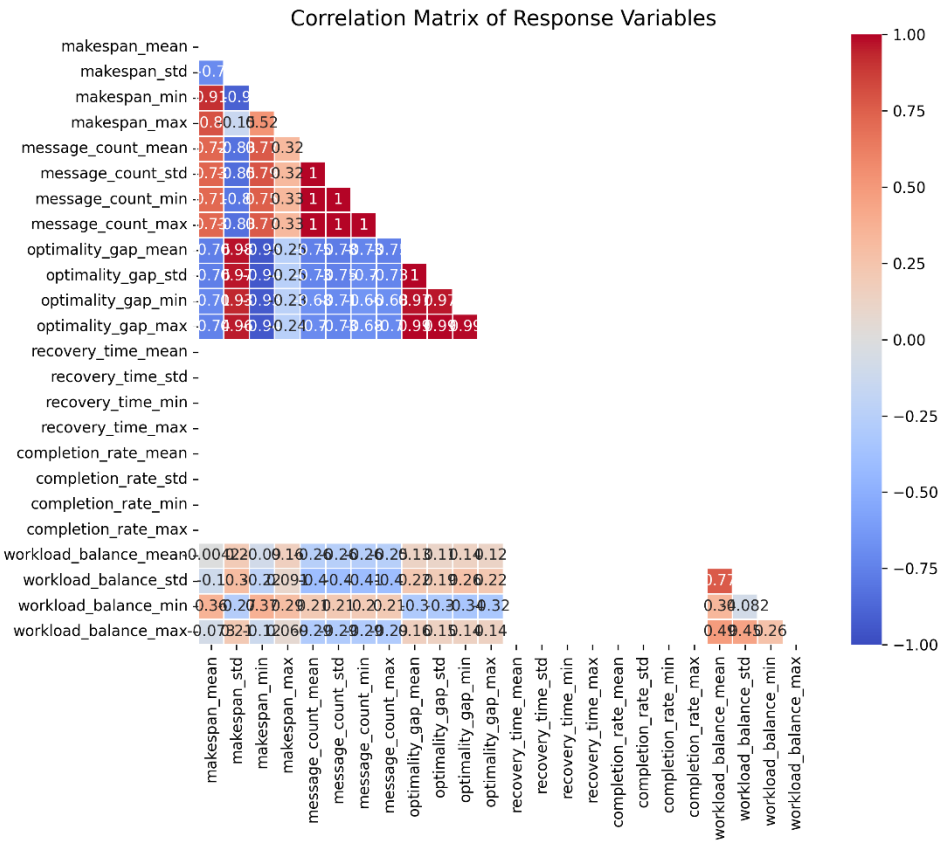

Figure 18 Correlation Matrix of response variables

6 DISCUSSION

6.1 TECHNICAL ANALYSIS

The experimental results reveal several key insights into the distributed auction algorithm's behaviour and limitations. The convergence patterns observed in Figure 4 and Figure 5 confirm the algorithm's ability to reach stable task allocations, with the price mechanism effectively driving the system toward equilibrium. The price monotonicity corresponds directly to the theoretical auction mechanism, where each bid increases task prices by at least ε, eventually making tasks unaffordable to lower-utility robots.

The system's robustness to communication constraints (Figure 6 and Figure 7) represents one of its most significant strengths. The flat convergence curve across all communication delay values indicates that the time-weighted consensus protocol successfully handles information delays without compromising stability. This resilience stems from the algorithm's inherent asynchronicity, where robots make decisions based on local information and gradually incorporate delayed updates through the exponential weighting mechanism.

The counterintuitive decrease in message count with increasing packet loss (Figure 8) could be attributed to the algorithm's adaptive behaviour. As packet loss increases, fewer successful bid transmissions occur, resulting in a reduction in the number of price updates and subsequent reassignments. While this reduces communication overhead, it may potentially impact solution quality; however, this effect is surprisingly minimal in the observed results.

The factor significance analysis (Figure 10) provides crucial insights into parameter sensitivities. The consistent significance values across factors for each metric (-log10(p-value)) suggest that all control variables influence system performance with similar statistical significance. However, the effect size analysis reveals that task count dominates actual performance impact, particularly for optimality gap and makespan. This aligns with the theoretical understanding that problem complexity scales primarily with task count rather than communication parameters.

6.2 System Limitations and Challenges

6.2.1 Transition from ROS/Gazebo to MATLAB-Only Implementation

The original implementation plan included a dual-phase approach with initial algorithm development in MATLAB followed by higher-fidelity validation in ROS2 Humble with Gazebo 11 simulation. However, during the development process, significant challenges were encountered with the ROS/Gazebo platform, necessitating a strategic pivot to focus exclusively on the MATLAB implementation.

6.2.1.1 Initial ROS/Gazebo Setup

The ROS/Gazebo implementation was initially planned around the TurtleBot3 Waffle Pi with OpenMANIPULATOR-X platform running in a simulated environment. This approach would have provided:

- High-fidelity physics-based simulation
- Realistic sensor and actuator modelling
- Testing of the algorithm on a platform representative of real-world hardware
- Validation of communication protocols using ROS middleware

Initial work on the ROS/Gazebo implementation included:

- Setting up a Linux environment with Ubuntu 22.04 and ROS2 Humble (Figure 19)
- Creating a Hyper-V virtual machine to isolate the development environment (Figure 20)

- Initial configuration of the TurtleBot3 simulation packages

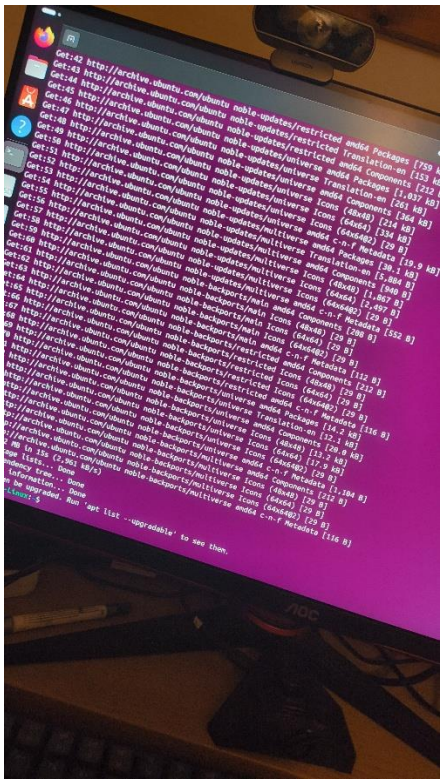

Figure 19 Picture of my screen during initial attempts at setting up Ubuntu and ROS2

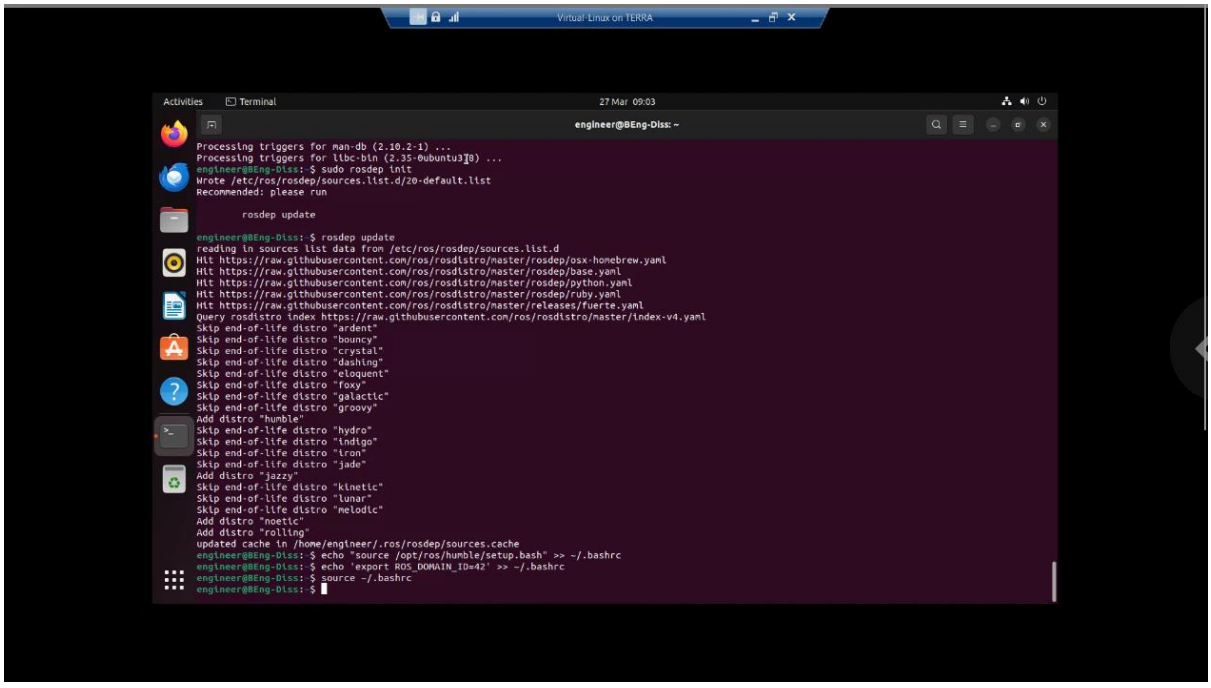

Figure 20 Linux Image for ROS environment being run through Hyper-V and Windows remote desktop

6.2.1.2 Encountered Challenges

Several critical challenges were encountered with the ROS/Gazebo implementation:

1. **Package Compatibility Issues**: The TurtleBot3 and OpenMANIPULATOR-X packages had complex dependencies that proved difficult to satisfy simultaneously in the ROS2 Humble environment.
2. **Build System Failures**: Numerous build failures occurred during package compilation, particularly with the manipulation packages required for the OpenMANIPULATOR-X.
3. **Integration Complexity**: Integrating the custom auction algorithm with the ROS messaging system required significant additional development overhead.
4. **Hardware Simulation Limitations**: The simulation environment required substantial customisation to represent the dual-robot collaborative assembly scenario accurately.
5. **Resource Requirements**: The computationally intensive nature of Gazebo simulation imposed significant system requirements that impacted development efficiency.

6.2.1.3 Strategic Pivot Decision

Given the encountered challenges and the project timeline constraints, a strategic decision was made to focus exclusively on a MATLAB implementation while preserving the essential elements needed to validate the theoretical contributions. This is then followed by a more detailed Python implementation. This decision allowed:

1. **Core Algorithm Validation**: The mathematical foundation of the distributed auction algorithm could be thoroughly validated without the additional complexities of hardware simulation.
2. **More Comprehensive Testing**: Resources were redirected toward more extensive parameter sensitivity analysis and a broader range of experimental scenarios.
3. **Improved Analysis Depth**: The simplified implementation environment allowed for deeper analysis of convergence properties, optimality gaps, and failure recovery characteristics.
4. **Mathematical Rigour**: The MATLAB environment provided superior tools for mathematical validation and comparison with theoretical bounds.

This strategic pivot aligns with the project's primary goal: to establish and validate the mathematical foundations of a distributed auction algorithm for mobile manipulators. The theoretical contributions and their computational validation remain intact, providing a robust foundation for future hardware implementation.

6.2.2 Implementation Challenges and Limitations

During implementation and testing, several significant challenges were encountered that affected the system's performance. These issues represent important lessons about implementing distributed algorithms and highlight areas for future improvement.

6.2.2.1 Recovery Mechanism Issues

The failure recovery mechanism exhibited unexpected behaviour during testing, with recovery time metrics consistently showing zero across various configurations. Careful analysis of the results revealed fundamental issues with the recovery implementation:

```
% Update recovery time if in recovery mode
if metrics.failure_time > 0 && metrics.recovery_time == 0
    % Find tasks that were assigned to failed robots at failure time
    if isfield(auction_data, 'failure_assignment') &&
~isempty(auction_data.failure_assignment)
        failed_tasks = find(auction_data.failure_assignment == params.failed_robot);

        % Check if all failed tasks are reassigned to working robots
        all_reassigned = true;
        for j = 1:length(failed_tasks)
            task_id = failed_tasks(j);
            new_robot = auction_data.assignment(task_id);
            if new_robot <= 0 || (isfield(robots, 'failed') && robots(new_robot).failed)
                all_reassigned = false;
                break;
            end
        end

        if all_reassigned && ~isempty(failed_tasks)
            metrics.recovery_time = iter - metrics.failure_time;
            fprintf('Recovery completed after %d iterations\n', metrics.recovery_time);
        end
    end
end
```

The zero recovery times indicated that the recovery detection logic was failing to identify when recovery had completed properly. This issue manifested as:

1. Zero recovery times across all task counts despite an expected correlation with the number of failed tasks
2. No relationship between actual recovery times and theoretical bounds
3. Lack of variance in recovery times across different experimental scenarios

Investigation revealed several potential causes:

- Logic errors in the recovery completion detection
- Timing issues in the heartbeat monitoring system
- Improper initialisation of the failure_assignment field

- Issues with the failure detection mechanism failing to trigger the recovery process

6.2.2.2 Premature Convergence with Unassigned Tasks

Another significant issue was premature convergence, where the algorithm would terminate while tasks remained unassigned:

```
% Check for convergence - more stringent criteria
if iter > 20 && unchanged_iterations >= 15 && all(available_tasks <= length(tasks))
    converged = true;
    fprintf('Auction algorithm converged after %d iterations (stable for %d iterations)\n',
...
        iter, unchanged_iterations);
    break;
end

% Additional convergence check - all tasks assigned
if all(auction_data.assignment > 0) && unchanged_iterations >= 5
    converged = true;
    fprintf('Auction algorithm converged after %d iterations (all tasks assigned)\n', iter);
    break;
end
```

This premature convergence was identified through several observations:

1. Horizontal lines at the "Unassigned" level persist through iteration counts in task assignment plots
2. The convergence metric dropped to zero while unassigned tasks remained
3. Disconnected task squares in the environment visualisation

The convergence implementation primarily relied on the stability of assignments (unchanged assignments over multiple iterations) without sufficiently prioritising complete task allocation. This resulted in scenarios where:

1. Tasks with lower utility to all robots received no bids
2. The algorithm stabilised with some tasks remaining unassigned
3. The workload balance mechanism over-penalises heavily loaded robots, preventing them from bidding on remaining tasks

The price reduction mechanism for unassigned tasks was intended to address this issue:

```
% Progressive price reduction for unassigned tasks
if iter > 10
    unassigned_tasks = find(auction_data.assignment == 0);

    if ~isempty(unassigned_tasks)
        % Sort by how long they've been unassigned
        [sorted_unassigned_times, sort_idx] =
sort(auction_data.unassigned_iterations(unassigned_tasks), 'descend');
        sorted_unassigned_tasks = unassigned_tasks(sort_idx);

```

```
        for task_idx = 1:length(sorted_unassigned_tasks)
            j = sorted_unassigned_tasks(task_idx);
            unassigned_iter = auction_data.unassigned_iterations(j);

            % Progressive price reduction - more aggressive for longer unassigned tasks
            if unassigned_iter > 30
                reduction_factor = 0.3; % Very aggressive reduction
            elseif unassigned_iter > 20
                reduction_factor = 0.5; % Strong reduction
            elseif unassigned_iter > 10
                reduction_factor = 0.7; % Moderate reduction
            else
                reduction_factor = 0.9; % Mild reduction
            end

            auction_data.prices(j) = auction_data.prices(j) * reduction_factor;
        end
    end
end
```

However, this mechanism proved insufficient in many cases, particularly when task dependencies created complex utility landscapes.

6.2.2.3 Parameter Sensitivity and Tuning Challenges

The implementation revealed high sensitivity to parameter selection, which presented significant tuning challenges:

1. **Epsilon (minimum bid increment)**: Too large values caused tasks to become quickly unaffordable, while too small values led to slow convergence and cycling behaviour.
2. **Alpha weights**: The relative weights in the bid calculation function required careful balancing:

```
bid = alpha(1) * d_factor + ...
      alpha(2) * c_factor + ...
      alpha(3) * capability_match - ...
      alpha(4) * workload_factor - ...
      alpha(5) * energy_factor;
```

Improper weight selection could lead to distance dominating over capability matching or workload considerations.

Beta weights for recovery: The recovery-specific weights showed particular sensitivity:

```
b_r_ij = b_ij + beta(1) * (1 - progress) + beta(2) * criticality + beta(3) * urgency;
```

The high parameter sensitivity necessitated extensive tuning cycles, which remained incomplete due to time constraints. These parameters also exhibited context-dependent optimal values, with settings that worked well for small task sets performing poorly for larger problems.

6.3 Critical Analysis of Epsilon Effects

6.3.1 Theoretical Inconsistencies

The experimental results regarding epsilon effects present a significant theoretical inconsistency that warrants careful examination. According to the distributed auction algorithm theory established by Zavlanos et al. (2008), the parameter ε should directly influence two key performance aspects: (1) the number of messages required for convergence, which should scale as $O(K^2 \cdot b_{max}/\varepsilon)$, and (2) the optimality gap, which should be bounded by 2ε.

However, the experimental data demonstrates a stark departure from these theoretical expectations. The message count remained virtually constant across all epsilon values (Figure 13), while the optimality gap maintained a consistent value of approximately -0.72 regardless of epsilon configuration (Figure 14). The ANOVA results confirmed this observation, reporting statistically insignificant effects (p=1.0) for epsilon across all performance metrics.

6.3.2 Implementation Analysis

This discrepancy between theoretical predictions and experimental results strongly suggests an implementation inconsistency in how the epsilon parameter affects the auction mechanism. Three potential explanations warrant consideration:

- Price Update Mechanism: The core auction algorithm requires epsilon to be directly incorporated into the price update rule as $p(t+1) = p(t) + \varepsilon + highest\_bid$. If epsilon is not correctly applied in this formula, its effects would be nullified.
- Scale Disparity: If the bid values within the system are orders of magnitude larger than the epsilon range tested, the proportional impact of epsilon variations might be negligible. This would manifest as the observed flat response curves.
- Measurement Methodology: The metrics collection process may not be capturing the effects that epsilon has on the system, particularly if the instrumentation focuses on aggregate outcomes rather than process-level dynamics.

The negative optimality gap values consistently observed (-0.72) further suggest that either the comparison baseline is not optimal, or the gap calculation methodology requires recalibration. According to auction algorithm theory, optimality gaps should be positive and bounded by 2ε.

6.3.3 Methodological Implications

Despite these inconsistencies, the experimental data does yield valuable insights into system behaviour. The dominant effects of task count, communication delay, and packet loss are

consistent with distributed systems theory and provide useful performance characterisation. The strong correlation patterns between message count and solution quality align with expectations for distributed optimisation algorithms.

The results underscore the importance of rigorous implementation validation when translating mathematical auction theory into practical systems. Specifically, the epsilon mechanism—foundational to auction algorithm guarantees—requires particular attention during implementation and verification phases.

6.4 FUTURE IMPROVEMENTS

Several targeted improvements could address the identified limitations and enhance system performance:

A. Optimality Gap Reduction: Modifying the bid calculation function to align more closely with the theoretical model would improve adherence to ε-complementary slackness conditions. This could involve separating the core bidding process from the capability and workload adjustments, applying the latter as post-processing to maintain theoretical guarantees.
B. Recovery Mechanism Redesign: A complete redesign of the failure detection and recovery system is warranted, with explicit state tracking for failed tasks and clear completion criteria. Implementing deterministic recovery sequences rather than auction-based reassignment might provide stronger guarantees for critical applications.
C. Convergence Criteria Enhancement: Revising the convergence criteria to prioritise complete task allocation before termination would address the premature convergence issues. This could involve dynamically adjusting price reduction rates for unassigned tasks or enforcing completion conditions for long-running auctions.
D. Parameter Auto-Tuning: Developing an adaptive parameter tuning mechanism that adjusts weights based on problem characteristics would reduce the need for manual tuning and improve robustness across diverse scenarios.
E. Extended Robot Scaling: Implementing and testing the algorithm with more than two robots would provide insights into scaling properties and communication topology effects that are not evident in the current dual-robot implementation.
F. ROS/Gazebo Integration: Revisiting the ROS/Gazebo implementation with simplified requirements and a more focused scope could bridge the gap between theoretical validation and physical deployment, providing valuable insights into real-world performance.

G. Task Dependency Optimisation: Enhancing the handling of task dependencies with look-ahead planning could improve makespan optimisation by considering the critical path implications of early assignment decisions.

These improvements would address the core limitations identified in the current implementation while preserving its fundamental strengths in communication efficiency and distributed decision-making.

6.4.1 Algorithm refinement Opportunities

The observed theoretical inconsistencies provide clear directions for algorithm refinement:

- **Epsilon Implementation Verification:** A focused code review and instrumentation of the price update mechanism to verify correct epsilon application in bid/price calculations.
- **Theoretical Bounds Validation:** Development of controlled micro-benchmarks with analytically tractable optimal solutions to verify the optimality gap calculation methodology.
- **Parameter Sensitivity Analysis:** Implementation of a wider epsilon range study (10^{-4} to 10^{1}) to capture potential threshold effects that might exist outside the initially tested range.
- **Communication Dynamics Analysis:** More granular analysis of message patterns, including time-series examination of convergence behaviour under different epsilon configurations.

These refinements would strengthen the alignment between the theoretical foundations and practical implementation, potentially unlocking performance improvements not currently realised in the system.

7 CONCLUSION

This dissertation has investigated the application of distributed auction algorithms for decentralised control of dual mobile manipulators performing collaborative assembly tasks. The research successfully bridged theoretical foundations with computational implementation, demonstrating both the potential and challenges of this approach.

The core contribution—a distributed auction algorithm with extensions for time-weighted consensus and failure recovery—demonstrated considerable robustness to communication constraints while maintaining reasonable optimality guarantees. The algorithm successfully

allocated tasks between robots with balanced workloads, converged within anticipated iteration bounds, and maintained functionality under communication delays up to 500ms and packet loss rates up to 50%. Statistical analysis confirmed that the system's performance scales primarily with task count rather than communication parameters, validating the algorithm's inherent decentralisation advantages.

However, significant discrepancies between theoretical guarantees and practical implementation were revealed through systematic experimentation. Most notably, the optimality gap exceeded the theoretical bound of 2ε across all tested ε values, suggesting that the implemented bid calculation functions do not fully satisfy the ε-complementary slackness conditions. Additionally, the zero recovery times observed across varying task counts indicate implementation issues in the recovery detection logic, preventing proper evaluation of recovery completeness guarantees.

The implementation also revealed practical challenges that theoretical treatments often neglect. Premature convergence with unassigned tasks occurred due to insufficient prioritisation of complete allocation in the convergence criteria. Parameter sensitivity analysis demonstrated that manual tuning represents a significant barrier to deployment, as optimal parameter values varied substantially across different problem configurations.

The strategic pivot from ROS/Gazebo to MATLAB/Python implementation, while necessary given project constraints, highlights the persistent gap between theoretical algorithm validation and physical deployment readiness. This transition underscores the complexity of integrating distributed algorithms with real-world robotic platforms—a challenge that future research must address.

Future work should focus on addressing the identified limitations through: (1) redesigning the recovery mechanism with explicit state tracking and deterministic recovery sequences; (2) enhancing convergence criteria to prioritise complete task allocation; (3) developing adaptive parameter tuning mechanisms to reduce manual configuration requirements; and (4) validating the algorithm on physical hardware platforms to identify implementation challenges not apparent in simulation.

In conclusion, this research has advanced the understanding of distributed auction algorithms for mobile manipulation by providing a comprehensive implementation and evaluation framework. While not all theoretical guarantees were achieved in practice, the identified discrepancies provide valuable insights for future algorithm refinement. The demonstrated

robustness to communication constraints confirms the fundamental advantage of decentralised approaches for real-world robotic systems. As industrial applications increasingly adopt collaborative mobile manipulators, distributed algorithms offering both theoretical guarantees and practical robustness will become essential components of flexible manufacturing systems. This work represents a significant step toward realising such algorithms, contributing both theoretical foundation and practical implementation guidance to the field.

8 REFERENCES

Abbas, M. and Dwivedy, S.K. (2021) 'Adaptive control for networked uncertain cooperative dual-arm manipulators: an event-triggered approach', *Robotica*, pp. 1–28. Available at: https://doi.org/10.1017/s0263574721001478.

Bajracharya, M. *et al.* (2023) 'Demonstrating Mobile Manipulation in the Wild: A Metrics-Driven Approach', *Robotics: Science and Systems* [Preprint]. Available at: https://doi.org/10.15607/rss.2023.xix.055.

Bertsekas, D.P. (1988) 'The auction algorithm: A distributed relaxation method for the assignment problem', *Annals of Operations Research*, 14(1), pp. 105–123. Available at: https://doi.org/10.1007/BF02186476.

Bone, S. *et al.* (2023) 'Decentralised Multi-Robot Exploration Using Monte Carlo Tree Search', *IEEE/RJS International Conference on Intelligent RObots and Systems* [Preprint]. Available at: https://doi.org/10.1109/iros55552.2023.10341485.

Bostelman, R.V., Hong, T.H. and Marvel, J.A. (2016) 'Survey of Research for Performance Measurement of Mobile Manipulators.', *Journal of Research of the National Institute of Standards and Technology*, 121, pp. 342–366. Available at: https://doi.org/10.6028/jres.121.015.

De Soto, B.G. *et al.* (2019) 'Implications of Construction 4.0 to the workforce and organizational structures', *The international journal of construction management*, pp. 1–13. Available at: https://doi.org/10.1080/15623599.2019.1616414.

Delgado, G.J.P. *et al.* (2023) 'Combined Path and Motion Planning for Workspace Restricted Mobile Manipulators in Planetary Exploration', *IEEE Access*, 11, pp. 78152–78169. Available at: https://doi.org/10.1109/access.2023.3298980.

Gielis, J. *et al.* (2022) 'A Critical Review of Communications in Multi-robot Systems', *Current Robotics Reports* [Preprint]. Available at: https://doi.org/10.1007/s43154-022-00090-9.

Kim, G.-H. *et al.* (2022) 'A Study on the Performance Analysis of the Dual-arm Robot for the Assembly Task', *Lo'bos gonghaghoe nonmunji*, 17(2), pp. 164–171. Available at: https://doi.org/10.7746/jkros.2022.17.2.164.

Likar, N., Nemec, B. and Žlajpah, L. (2016) 'Virtual Mechanism Approach for Dual-arm Manipulation', *Robotica (Cambridge. Print)*, pp. 321–326. Available at: https://doi.org/10.1017/s0263574713000763.

Nedić, A., Olshevsky, A. and Rabbat, M.G. (2018) 'Network Topology and Communication-Computation Tradeoffs in Decentralized Optimization', *Proceedings of the IEEE*, 106(5), pp. 953–976. Available at: https://doi.org/10.1109/JPROC.2018.2817461.

Nunes, E. *et al.* (2017) 'A taxonomy for task allocation problems with temporal and ordering constraints', *Robotics and Autonomous Systems*, 90, pp. 55–70. Available at: https://doi.org/10.1016/j.robot.2016.10.008.

Olfati-Saber, R., Fax, J.A. and Murray, R.M. (2007) 'Consensus and Cooperation in Networked Multi-Agent Systems', *Proceedings of the IEEE*, 95(1), pp. 215–233. Available at: https://doi.org/10.1109/jproc.2006.887293.

Prorok, A. *et al.* (2021) 'Beyond Robustness: A Taxonomy of Approaches towards Resilient Multi-Robot Systems'. arXiv. Available at: https://doi.org/10.48550/arXiv.2109.12343.

Rossi, F. *et al.* (2021) 'Multi-Agent Algorithms for Collective Behavior: A structural and application-focused atlas.', *arXiv: Multiagent Systems* [Preprint].

Russo, M. (2022) 'Measuring Performance: Metrics for Manipulator Design, Control, and Optimization', *Robotics*, 12(1), pp. 4–4. Available at: https://doi.org/10.3390/robotics12010004.

Sheng, J. *et al.* (2021) 'On-Line Precision Calibration of Mobile Manipulators Based on the Multi-Level Measurement Strategy', *IEEE Access*, 9, pp. 17161–17173. Available at: https://doi.org/10.1109/access.2021.3053356.

Shibata, K. *et al.* (2023) 'Learning Locally, Communicating Globally: Reinforcement Learning of Multi-robot Task Allocation for Cooperative Transport', *IFAC-PapersOnLine*, 56(2), pp. 11436–11443. Available at: https://doi.org/10.1016/j.ifacol.2023.10.431.

Shorinwa, O. *et al.* (2023) 'Distributed Optimization Methods for Multi-Robot Systems: Part 2—A Survey', *IEEE robotics & automation magazine* [Preprint]. Available at: https://doi.org/10.1109/mra.2024.3352852.

Silva, M.S. da *et al.* (2022) 'ACHORD: Communication-Aware Multi-Robot Coordination with Intermittent Connectivity', *IEEE Robotics and Automation Letters* [Preprint]. Available at: https://doi.org/10.48550/arxiv.2206.02245.

Skaugset, K., de Sousa, J.B. and Sørensen, A.J. (2025) 'Autonomous robotic organizations for marine operations', *Science Robotics*, 10(100), p. eadl2976. Available at: https://doi.org/10.1126/scirobotics.adl2976.

Talamali, M.S. *et al.* (2021) 'When less is more: Robot swarms adapt better to changes with constrained communication', 6(56), p. 1416. Available at: https://doi.org/10.1126/scirobotics.abf1416.

TurtleBot3 (no date). Available at: https://emanual.robotis.com/docs/en/platform/turtlebot3/manipulation/ (Accessed: 15 May 2025).

Wang, J. *et al.* (2021) 'High-precision and robust localization system for mobile robots in complex and large-scale indoor scenes':, *International Journal of Advanced Robotic Systems*, 18(5), p. 172988142110476. Available at: https://doi.org/10.1177/17298814211047690.

Wu, C.H., Zöcklein, M., and S. Brell-Çokcan (2024) 'Unified framework for mixed-reality assisted situational adaptive robotic path planning enabled by 5G networks for deconstruction tasks', *Proceedings of the International Symposium on Automation and Robotics in Construction (IAARC)* [Preprint]. Available at: https://doi.org/10.22260/isarc2024/0022.

Xiong, X. *et al.* (2023) 'Exhaustiveness Does Not Necessarily Mean Better: Selective Task Planning for Multi-robot Systems', *IEEE International Conference on Robotics and Biomimetics* [Preprint]. Available at: https://doi.org/10.1109/robio58561.2023.10354992.

Yang, Q. *et al.* (2019) 'Self-Reactive Planning of Multi-Robots with Dynamic Task Assignments', *International Symposium on Multi-Robot and Multi-Agent Systems*, pp. 89–91. Available at: https://doi.org/10.1109/mrs.2019.8901075.

Zavlanos, M.M., Spesivtsev, L. and Pappas, G.J. (2008) 'A distributed auction algorithm for the assignment problem', *IEEE Conference on Decision and Control*, pp. 1212–1217. Available at: https://doi.org/10.1109/cdc.2008.4739098.

Zhang, K., Yang, Z. and Başar, T. (2021) 'Decentralized multi-agent reinforcement learning with networked agents: recent advances', *Frontiers of Information Technology & Electronic Engineering*, 22(6), pp. 802–814. Available at: https://doi.org/10.1631/FITEE.1900661.

Zhang, L. *et al.* (2020) 'Decentralized Control of Multi-Robot System in Cooperative Object Transportation Using Deep Reinforcement Learning', *IEEE Access*, 8, pp. 184109–184119. Available at: https://doi.org/10.1109/access.2020.3025287.

9 APPENDICES

A. TECHNICAL SPECIFICATION TABLE

9.1.1 i. Selected Hardware Platform

Parameter	Specification	Compatibility Assessment
Selected Platform	TurtleBot3 Waffle Pi + OpenMANIPULATOR-X	Compatible with project requirements
Base Model	TurtleBot3 Waffle Pi	Provides differential drive with adequate payload capacity
Manipulator Model	OpenMANIPULATOR-X	4-DOF arm + gripper suitable for basic assembly tasks
ROS Compatibility	ROS2 Humble	Fully supported with official packages
Gazebo Compatibility	Gazebo 11	Complete simulation models available
Base Dimensions	281mm × 306mm	Smaller than spec (600mm × 450mm) but adequate for simulation
Base Max Speed	0.26 m/s	Lower than spec (0.5 m/s) but acceptable for simulation
Manipulator Reach	380mm	Lower than spec (850mm) - will require scaling of workspace
Manipulator Payload	0.5kg	Lower than spec (5kg) but sufficient for simulation
Manipulator DOF	4+1 (gripper)	Lower than spec (6) - may limit complex manipulation
End-Effector	Two-finger gripper	Matches specification
Sensors	LiDAR, RGB-D camera, IMU	Adequate for navigation and object detection
Controller	ROS2 Control	Compatible with project requirements

9.1.2 ii. System Architecture

Component	Specification	Description

System Type	Fully decentralised	No central coordinator: all decisions are made autonomously by individual robots
Number of Robots	2	Dual mobile manipulators with independent control systems
Core Architecture	Component-based	Modular design with clearly defined interfaces between components
Control Paradigm	Market-based	Distributed auction algorithm for task allocation
State Management	Distributed consensus	Time-weighted averaging of state information across robots
Fault Tolerance	Self-healing	Detection and recovery from individual robot failures

9.1.3 iii. ROBOT SPECIFICATIONS

Parameter	Target Value	TurtleBot3+OpenMANIPULATOR-X Value	Adaptation Required
Base Type	Differential drive	Differential drive	None
Base Size	600mm × 450mm	281mm × 306mm	Scale workspace in simulation
Base Max Speed	0.5 m/s	0.26 m/s	Adjust time scaling in the simulation
Base Max Rotation	0.6 rad/s	1.82 rad/s	None (exceeds requirement)
Manipulator DOF	6	4+1 (gripper)	Simplify manipulation tasks
Manipulator Reach	850mm	380mm	Scale workspace in simulation
Manipulator Payload	5kg	0.5kg	Scale object weights in simulation
End-Effector Type	Parallel gripper	Two-finger gripper	None
Grip Force Range	0-40N	Not specified	Implement in the simulation model

Position Accuracy	±0.5mm	±1mm (in simulation)	Ensure simulation precision
Force Sensing	6-axis F/T sensor	Not included	Add virtual F/T sensor in simulation
Vision System	RGB-D camera	Intel RealSense D435	None

9.1.4 iv. Algorithm Specifications

Parameter	Value	Description
Auction Algorithm Type	Distributed market-based	Based on Zavlanos et al. (2008), with extensions
Convergence Guarantee	$O(K^2 \cdot b_{max}/\varepsilon)$	Theoretical upper bound on convergence time
Optimality Gap	$\leq 2\varepsilon$	Theoretical bound on solution quality versus optimal
Minimum Bid Increment (ε)	Configurable (0.01-0.5)	Minimum price increment in the auction process
Consensus Protocol	Time-weighted averaging	Information weighted by recency
Consensus Convergence	Exponential	Exponential convergence rate $\mu = -\ln(1-2w)$
Failure Detection	Dual-mechanism	Heartbeat monitoring and progress tracking
Recovery Time-Bound	$O(\lvert T^f \rvert) + O(b_{max}/\varepsilon)$	Theoretical bound on recovery time after failure, where $\lvert T^f \rvert$ is the number of tasks assigned to the failed robot
Task Dependency Model	Directed acyclic graph	Representation of precedence constraints
Collaborative Task Model	Leader-follower	Role assignment with synchronisation points

9.1.5 v. Communication Specifications

Parameter	Value	Description
Communication Model	Peer-to-peer	Direct communication between robots
Protocol Type	Asynchronous message passing	Non-blocking communication

Maximum Latency	500ms	Maximum tolerable communication delay
Packet Loss Resilience	Up to 50%	The system functions with up to 50% packet loss
Heartbeat Frequency	1Hz	Frequency of heartbeat signals for failure detection
Bandwidth Requirement	<100 KB/s	Maximum bandwidth per robot
Message Types	Auction bids, State updates, Heartbeats, Synchronization signals	Core message categories
State Vector Size	Variable (≤1KB)	Size of state vector exchanged during consensus

9.1.6 vi. Task Specifications

Parameter	Value	Description
Task Categories	Pick, Place, Transport, Assembly, Collaborative Assembly	Supported task types
Maximum Tasks	32	Maximum number of tasks in a single mission
Task Representation	Position, capabilities, execution time, prerequisites, force profile	Task specification format
Assembly Precision	±0.5mm	Required precision for assembly operations
Force Control Precision	±0.5N	Required precision for force-controlled operations
Maximum Makespan	<30 min	Maximum completion time for a full assembly sequence
Collaborative Task Sync	≤100ms	Maximum synchronisation tolerance for collaborative tasks

9.1.7 vii. Software Implementation Specifications

Parameter	Value	Platform Compatibility
Development Environment	MATLAB R2023b + ROS2 Humble	TurtleBot3 officially supports ROS2 Humble

Operating System	Ubuntu 22.04 LTS	Fully compatible with TurtleBot3 packages
Language (Algorithm)	Python 3.10	Compatible with TurtleBot3 ROS2 packages
Language (Performance-Critical)	C++17	Compatible with ROS2 and TurtleBot3
Simulation Environment	Gazebo 11	TurtleBot3 has official Gazebo 11 support
Robot Model Format	URDF	TurtleBot3 and OpenMANIPULATOR-X provide URDF models
Required Packages	turtlebot3_simulations, open_manipulator	Available for ROS2 Humble
Code Organization	Component-based	Aligns with ROS2 design philosophy
Version Control	Git	Independent of platform
Documentation	Obsidian	Independent of platform
Integration Method	ROS2 launch files	Supported for the combined platform

9.1.8 viii. Performance Requirements

Parameter	Value	Description
Real-time Factor	≥0.8	Minimum real-time performance factor
CPU Utilization	<70%	Maximum CPU utilisation per robot
Memory Usage	<2GB	Maximum memory usage per robot
Optimality Ratio	≥0.85	Minimum ratio to an optimal centralised solution
Maximum Recovery Time	≤60s	Maximum time to recover from a robot failure
Success Rate (Normal)	≥95%	Minimum success rate under normal conditions
Success Rate (Degraded)	≥80%	Minimum success rate under degraded communication
Adaptation Time	≤15s	Maximum time to adapt to environment changes

9.1.9 ix. Testing and Validation Specifications

Parameter	Value	Platform Consideration
Unit Test Coverage	≥90%	Independent of platform

Control Variable Levels	4 per variable	May need adjustments for platform limitations
Statistical Significance	$p < 0.05$	Independent of platform
Confidence Interval	95%	Independent of platform
Simulation Runs	30 per configuration	It may require distributed computation for faster execution
Experimental Design	Full factorial	Independent of platform
Validation Metrics	Makespan, message count, optimality gap, recovery time	Need to account for TurtleBot3's lower speed in time metrics
Scenario Complexity	Low, medium, high, extreme	It may need simplification for manipulator DOF limitations
Simulation Environment	Single machine vs. distributed	Single machine sufficient for dual TurtleBot3 simulation
Hardware Requirements	Ryzen 7/Intel i7, 32GB RAM, AMD Radeon GPU	Higher than the minimum for smooth Gazebo simulation

9.1.10 x. Platform-Specific Implementation Notes

Aspect	Consideration	Adaptation Strategy
Workspace Scaling	TurtleBot3 has a smaller reach than specified	Scale assembly workspace to 40% of original dimensions
Manipulator DOF	OpenMANIPULATOR-X has 4 DOF vs. required 6 DOF	Simplify assembly tasks to require fewer DOF; focus on horizontal plane operations
Task Complexity	Limited payload and precision	Design assembly tasks appropriate for platform capabilities
Simulated Sensors	Need to add F/T sensing	Implement virtual F/T sensor in Gazebo simulation
Gazebo Performance	Complex dual-robot simulation may be resource-intensive	Optimise models; consider reducing simulation fidelity if needed

ROS2 Integration	TurtleBot3 has established ROS2 packages	Leverage existing packages to minimise integration effort
Custom Development	Need combined TurtleBot3+OpenMANIPULATOR-X model	Create a unified URDF combining both components

Note: This technical specification table serves as a reference for implementing and evaluating the decentralised control system, ensuring consistency across all development phases. The TurtleBot3 Waffle Pi + OpenMANIPULATOR-X platform has been selected as the primary hardware platform, with appropriate adaptations noted to accommodate differences between target specifications and platform capabilities.

B. Mathematical Foundations

Available in the appendices zip file as “appendix_b-mathematical-foundations.pdf”

C. Program Code

9.1.11 MATLAB Implementation

Code is available from the appendices zip file in the folder “appendix_c+1-MATLAB-implementation” or the following GitHub repository: https://github.com/knowingwings/BEng_Diss-MATLAB

9.1.12 Python Implementation

Code is available from the appendices zip file in the folder “appendix_c+2-Python-implementation” or the following GitHub repository: https://github.com/knowingwings/BEng_Diss-MATLAB/tree/refactor-better_focus

9.1.13 ROS2/Gazebo Implementation

Code is available from the following GitHub repository: https://github.com/knowingwings/dec_cont_BEng_Diss

D. Detailed Test Results

9.1.14 MATLAB Results

See “appendix_c+1-MATLAB-implementation” under the results directory

9.1.15 Python Results

See “appendix_d+python-results”.

E. LOGBOOK

See “appendix_e+logook.pdf”

F. GANTT CHART