

Comparative Analysis of Priority-based Sequential and Thread-based Concurrent Task Execution Strategies for Multi-Robot Warehouse Automation

Siddhartha Gupta

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
siddharthagupta01@gmail.com*

Sanjog Sapkota

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
sanjogspk88@gmail.com*

Amrit Kumar Banjade

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
amritbanjade05@gmail.com*

Sumit Sigdel

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
sumsigdel@gmail.com*

Laxmi Prasad Upadhyaya

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
laxmiprasadupadhyaya0@gmail.com*

Nabin Lamichhane

*Department of Electronics and Computer Engineering
Pashchimanchal Campus, IOE, TU.
Pokhara, Nepal
babunabin@wrc.edu.np*

Abstract—A multi-robot system can improve time efficiency in warehouse transportation; however, some applications require predictability in addition to efficiency. The task execution strategy selected for a multi-robot system significantly influences task completion time, predictability, and system complexity. This paper presents an empirical study comparing two task execution strategies for a multi-robot warehouse platform: (1) a priority-based sequential approach, where a central controller runs a single thread to advance each robot's state individually, and (2) a thread-based concurrent approach, where each robot is managed by its own thread and coordinates through periodic checks of shared pose data. Both were tested against a single-robot baseline. Each robot is built with an ESP32 microcontroller, encoders for motion feedback, and a servo-driven gripping mechanism. Robots operate in a controlled platform that consists of drop zones and loads. The whole system uses ArUco marker-based overhead localization and a ROS 2 central controller communicating with each robot over UDP/JSON. Across the repeated trials of both task execution strategies on the system with varying load counts, the thread-based concurrent execution strategy finished tasks 27–49% faster but had 39–78% more timing unpredictability, compared to the priority-based sequential task execution strategy.

Keywords—ArUco marker localization, concurrent execution, ESP32, multi-robot systems, sequential execution, task execution strategy, timing predictability, UDP/JSON communication, warehouse robotics.

I. INTRODUCTION

Warehouse operations across the globe still rely heavily on manual labor for inventory handling, order picking and goods transport. Such reliance on manual processes leads to slow

throughput, high error rates, rising labor costs and pressures that have only been exacerbated with the rise of e-commerce and ongoing labor shortages across the logistics sector. Multi-agent robotic systems offer a promising path toward addressing these challenges, since deploying multiple robots in parallel raises the possibility of significantly higher throughput than manual or single-robot operations.

Realizing this potential, however, is not automatic. Simply adding more robots to a shared warehouse does not guarantee proportional gains in performance. As agents operate concurrently, they must contend for shared resources, physical space, and task assignments, and the execution strategy governing how tasks are scheduled and dispatched.

This research investigates that relationship directly by comparing priority-based sequential execution against thread-based concurrent execution within a multi-agent warehouse system. Rather than proposing a new coordination algorithm, this work focuses on systematically evaluating how these execution strategies perform against one another and on measuring not just how fast each approach completes work, but how predictable that performance is across repeated trials.

To ground this comparison in a realistic setting, the evaluation is conducted on a multi-agent platform built from affordable, open-source hardware and software components, reflecting the kind of system that would be practically deployable for small and medium-sized warehouse operations rather than requiring large-scale commercial infrastructure. Reliable

coordination in this setting depends on accurate localization: single-sensor methods relying solely on wheel encoders are prone to drift and wheel slippage, while methods relying solely on a single onboard camera are vulnerable to occlusion in cluttered environments. The platform addresses this through sensor fusion, combining continuous encoder-based odometry with periodic, drift-free position correction from an overhead-camera ArUco marker system, giving each robot a reliable pose estimate without dependence on expensive dedicated hardware. This localization approach provides the foundation on which the sequential and concurrent execution strategies are evaluated.

II. LITERATURE REVIEW

Parallel task execution is one of the primary motivations for using multi-agents. The tasks which are independent or can be decomposed into concurrently executable subtasks, they can be assigned to different agents hence performing the work simultaneously. This potentially reduces overall execution time. Miloradović *et al.* worked on parallel task execution for multi-agent mission planning and formulated the problem using Integer Linear Programming and Constraint Programming and demonstrated how parallel execution can be incorporated while considering task relationships and execution constraints [1]. Agrawal *et al.* investigated multi-robot task allocation in warehouse environments using an attention-inspired reinforcement learning approach [8]. Krnjaic *et al.* used a hierarchical manager-worker MARL architecture to assign goals across robotic and human co-workers, scaling across diverse warehouse configurations [9].

Multi-agent path finding studies indicate that concurrent operation can improve system efficiency, but its benefits depend on how effectively the conflict and interactions among the agents are managed. Fredriksson *et al.* proposed a path finding technique using Conflict-Based Search (CBS) and structural semantic topometric maps [7].

Ghassemi and Chowdhury tackled decentralized task allocation in disaster response by introducing online algorithms that use weighted bipartite matching [3]. These methods handle asynchronous task arrivals, deadlines, and heterogeneous robot capabilities (e.g., payload and range constraints), achieving near-optimal task completion rates and demonstrating scalability in dynamic, real-time scenarios. Kato and Kamoshida proposed a self-contained multi-agent architecture for logistics warehouse simulation to cut the cost and effort of building and modifying the warehouse simulation models. Based on their evaluation, they achieved a 31.5% reduction in development cost in comparison with a conventional multi-agent simulation environment (NetLogo) and better modifiability in some realistic warehouse reconfiguration scenarios [4].

When multiple agents work simultaneously, they may interfere with each other, requiring mechanisms for communication, prioritization or conflict resolution. Separately optimizing task allocation and path planning can produce inefficient coordination. Integrating both decisions into a unified process can improve scheduling efficiency and operational costs [6].

Pecora *et al.* proposed a loosely-coupled approach for multi-agent coordination allowing robots to perform planning and control independently while using coordination mechanisms to manage interaction among them [2]. They demonstrated concurrent operation does not necessarily require a complete centralized control; however, a coordination mechanism is required when agents interact. Yin *et al.* investigated cooperative path planning using asynchronous multi-agent reinforcement learning. They allowed agents to make progress without requiring strict synchronization among agents [10]. Similarly, Bowman *et al.* studied coordination of agents through asynchronous cloud communication and investigated the mechanisms for coordinating agents without requiring continuous communication [11]. Chiva Gil *et al.* introduced a predictability term, derived from free-energy minimization, that penalizes agents for deviating from what other agents predict they will do, allowing independent, sequential planners to coordinate resolving conflicts and eliminating deadlock without explicit communication [14].

The number of agents is an important factor in determining whether concurrent execution provides a performance advantage. Fredriksson *et al.* showed that an increasing number of agents can increase the number of potential conflicts [7]. Scalability, communication, and coordination are major challenges in systems containing multiple autonomous agents [5].

Concurrent execution may not always outperform sequential execution. Under some conditions, it is preferable to execute remaining work sequentially rather than continue it concurrently [12]. Oates and Bercher investigated sequential, parallel, partial-order, and partial-order causal-link plans and evaluated their effects on makespan and demonstrated that task dependencies and representation of plans influence the amount of useful parallelism that can be achieved [13].

Existing studies demonstrate that parallel execution can reduce execution time with sufficient independent work available. Most of them focus on developing scheduling, coordination and path planning mechanisms rather than directly comparing sequential and concurrent execution under controlled conditions. To address this gap, this study conducts a controlled comparative analysis of sequential and concurrent execution approaches in a multi-agent system for warehouse automation. The study evaluates their relative performance under the different load counts. In addition to performance, the study also looks at how predictable each approach is—whether completion time stays fairly consistent across trials or swings from run to run to capture reliability alongside average speed.

III. SYSTEM DESIGN AND METHODOLOGY

A. Hardware Design and Fabrication

The robots are differential-drive models with two powered wheels and a free-moving caster wheel for balance.

Each robot uses an ESP32-based controller, N20 DC motors with encoders, an L293D dual H-bridge motor driver, a Li-Po battery, and a servo motor for gripping loads. The encoders monitor wheel movement and help the robot stay on course by correcting differences between the drive wheels. The robots

connect wirelessly to the central controller using ESP32 Wi-Fi and UDP.

All electronics are mounted on a custom double-sided PCB that holds the microcontroller, motor driver, and other parts in a compact layout to fit the robot's chassis. Each robot has an ArUco marker on top so the central controller can identify it from above. Figure 1 shows the complete robot.

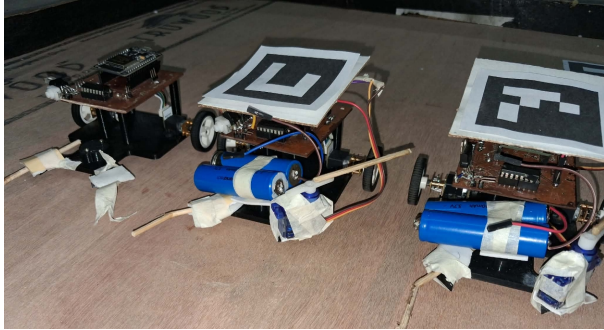


Fig. 1. Bot Construction

Both task execution strategies in this paper use the same hardware platform. The following comparison focuses only on the software that assigns tasks to the hardware, not on the hardware itself.

B. System Design and Navigation

The system utilizes a centralized control setup with an overhead camera, a ROS 2-based central controller, and mobile robots with ESP32 processors. The overhead camera is used to capture the workspace, and an ArUco detection node finds the markers placed on the robots, their loads, and their drop zones.

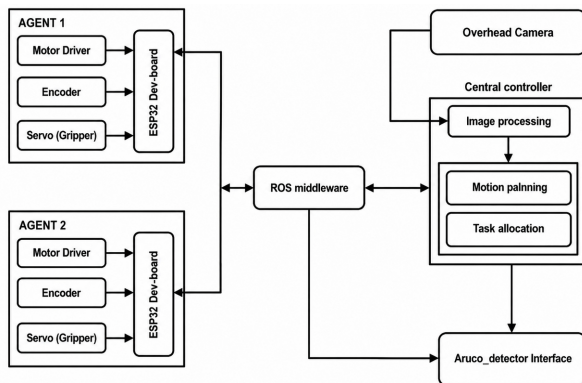


Fig. 2. Block Diagram of System

The navigation system used encoder feedback, PID control, and ArUco marker localization. Encoders allowed the robot to move smoothly accounting for the motor spin differences. The controller utilized captured video from the overhead camera to determine the direction and distance that the robot should travel. Each coordination system used the same methods for navigation, PID control, and motor speeds to ensure a fair comparison between the two algorithms.

C. ROS Architecture and Communication

The central controller is deployed on a PC running the ROS 2 framework, serving as the central hub for communication, task management, and coordination of robots in the warehouse.

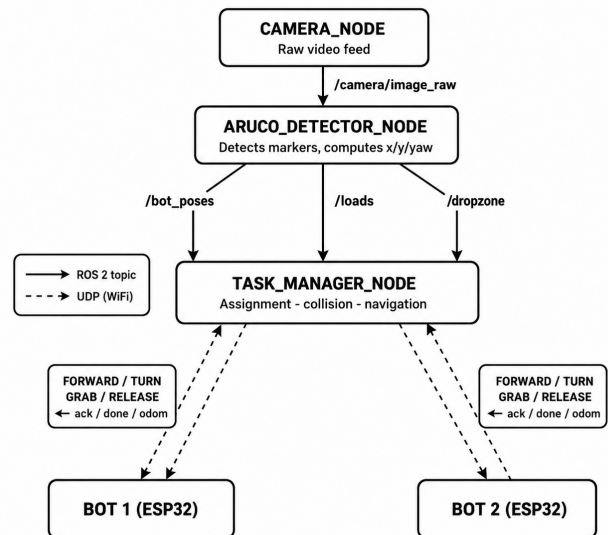


Fig. 3. ROS Architecture

1. *Camera Node*: The camera node takes the raw video frames from the camera and pushes them into the ROS 2 system through the `/camera/image_raw` topic. Every other node that exists in this research depends upon the information from this node in order to know where something is in the arena.
2. *ArUco Detector Node*: This node subscribes to the `/camera/image_raw` topic from the camera node and allows it to process every single video frame from the camera. This node detects ArUco markers and calculates the pose of bots and loads which includes relative X, Y and yaw values with respect to the arena. Finally, this node publishes this information onto three different topics for use by the other nodes in the research.
 - `/bot_poses`: The x and y positions of each bot, as well as the yaw angle of each bot
 - `/loads`: The x and y positions of each load in the arena
 - `/dropzone`: The x and y positions of the drop zone target for the loads
3. *Task Manager Node*: This node subscribes to the three topics published by the aruco_detector node. This node uses the information published from those topics to manage the tasks that each bot will perform in the arena.

The robots and the controller both use ROS 2 topics and UDP communication over Wi-Fi. The controller sends JSON messages to the robots with commands such as FORWARD, BACKWARD, TURN_L, TURN_R, GRAB, and RELEASE.

The robots then send back acknowledgment messages to the controller upon receiving and completing the sent commands.

D. Task Execution Strategies

1. **Priority-Based Sequential Execution:** In this approach both robots were operated within a single loop in which there was a predetermined priority level (Robot 1 > Robot 2) to determine which robot should move first. Both robots were advanced within the same 0.2s tick; motion commands were issued sequentially within that tick: Bot 1's command was always computed and issued first, and Bot 2's command was computed and issued only afterward, conditional on the proximity check to Bot 1. This ordered, priority-driven dispatch is what we refer to as *sequential* execution.

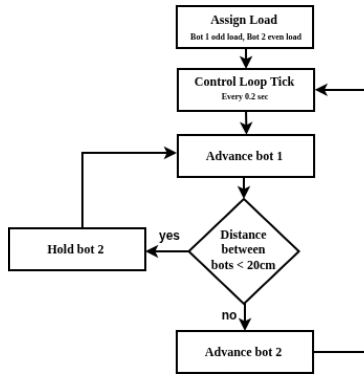


Fig. 4. Priority-Based Sequential Task Execution

2. **Thread-Based Execution:** Unlike the priority-based approach, each robot's control loop runs on an independent operating system thread (*threading.Thread*). Each of these threads runs concurrently, meaning that both robots move simultaneously. Each thread periodically reads the other robot's pose (updated via the ArUco localization system) to ensure the two robots do not collide with each other whenever the distance to the other robot falls below 20 cm.

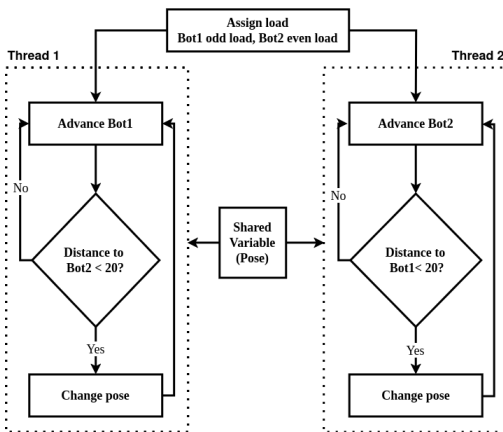


Fig. 5. Thread-Based Concurrent Task Execution

E. Experimental Setup

The presented experimental setup consists of plywood of size (91*93) cm, which represents a minimal warehouse platform. There are two drop zones placed at two separate corners at a distance of 70 cm. The size of the load and both robots are (6*6*6) cm and (12*10*8) cm, respectively. The camera is placed approximately 180 cm from the platform. Initially, both robots are placed at any corner with a drop zone, and loads are placed approximately 60 cm away from the robots. This setup is standard for our testing and remains the same for both execution strategies for variable load count.



Fig. 6. Experimental Setup

F. Evaluation Metrics

The performance of the priority-based sequential and thread-based concurrent task execution strategies was evaluated using a two-dimensional assessment that included both task completion time and variance in those times.

1. **Task Completion Time:** The time taken by the robot to reach, pick up, and drop the load at the specified drop zone. We performed tests up to 4 iterations across two, three, and four load counts, and calculated the average task completion time for each load count. The formula used to calculate average completion time was:

$$\bar{T} = \frac{1}{n} \sum_{i=1}^n T_i \quad (1)$$

where T_i is the completion time of the i -th trial and n is the total number of trials. The relative change in completion time of the thread-based strategy compared to the priority-based strategy, at a given load count, is calculated as:

$$\text{Time Difference (\%)} = \frac{|\bar{T}_P - \bar{T}_T|}{\max(\bar{T}_P, \bar{T}_T)} \times 100 \quad (2)$$

where $\bar{T}_P$ and $\bar{T}_T$ are the average completion times of the priority-based and thread-based strategies, respectively.

2. *Timing Variance*: The standard deviation and Coefficient of Variation (CV) of completion time were computed for each strategy at each load count to enable the characterization of scheduling predictability independent of raw speed.

$$CV = \frac{\sigma}{\bar{T}} \times 100 \quad (3)$$

where

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (T_i - \bar{T})^2} \quad (4)$$

T_i = Completion Time of the i -th Trial.

$\bar{T}$ = Average Completion Time.

A lower CV indicates that timing is consistent and more predictable. The relative change in CV between the two strategies was calculated at each load count using:

$$CV \text{ Difference (\%)} = \frac{|CV_T - CV_P|}{\max(CV_T, CV_P)} \times 100 \quad (5)$$

where CV_P and CV_T are the coefficients of variation of the priority-based and thread-based strategies, respectively.

3. *t-test*: To formally test whether the difference in mean completion time between the priority-based and thread-based strategies was statistically significant at each load count, Welch's two-sample t-test was used:

$$t = \frac{\bar{T}_P - \bar{T}_T}{\sqrt{\frac{s_P^2}{n_P} + \frac{s_T^2}{n_T}}} \quad (6)$$

where $\bar{T}_P$, $\bar{T}_T$ are the mean completion times and s_P^2 , s_T^2 are the sample variances of the priority-based and thread-based strategies, respectively, and $n_P = n_T = 4$. Welch's variant was chosen over the standard Student's t -test because it does not assume equal variance between groups, which is appropriate here given the substantial difference in CV observed between the two strategies (Tables II–III). The corresponding degrees of freedom were computed using the Welch–Satterthwaite equation:

$$df = \frac{\left(\frac{s_P^2}{n_P} + \frac{s_T^2}{n_T}\right)^2}{\frac{(s_P^2/n_P)^2}{n_P - 1} + \frac{(s_T^2/n_T)^2}{n_T - 1}} \quad (7)$$

A two-tailed p -value was then obtained from the t -distribution with the computed degrees of freedom; $p < .05$ was taken as the threshold for statistical significance.

4. *Confidence Interval*: To assess whether observed differences in mean completion time were statistically meaningful given the limited sample size ($n = 4$), a 95% confidence interval was computed for each mean using

the t -distribution rather than the normal distribution, appropriate for small samples:

$$CI = \bar{T} \pm t_{\alpha/2, n-1} \cdot \frac{\sigma}{\sqrt{n}} \quad (8)$$

where $t_{0.025, 3} = 3.182$ for four trials (3 degrees of freedom). Non-overlapping intervals between two strategies at a given load count indicate that the difference in means is unlikely to be due to random trial-to-trial variation alone.

IV. RESULTS AND ANALYSIS

A. Single-Robot Baseline

TABLE I
SINGLE-ROBOT COMPLETION TIME

Loads	Trial 1	Trial 2	Trial 3	Trial 4	Mean (s)	CV (%)	95% CI
2	40.08	38.37	37.67	36.45	38.14	3.98	[35.72, 40.56]
3	60.38	58.42	59.18	58.03	59.00	1.75	[57.35, 60.65]
4	70.13	78.34	75.24	77.35	75.27	4.86	[69.45, 81.09]

We also measured single-robot data to provide us with a baseline reference. As the number of loads increased, the completion time increased; the completion time nearly doubled as the number of loads doubled while the timing variance stayed low (CV 1.75%–4.86%) across all load counts because there was no coordination overhead.

B. Priority-Based Sequential Task Execution

TABLE II
PRIORITY-BASED SEQUENTIAL COMPLETION TIME

Loads	Trial 1	Trial 2	Trial 3	Trial 4	Mean (s)	CV (%)	95% CI
2	30.01	31.22	27.48	33.06	30.44	7.68	[26.72, 34.16]
3	39.37	38.56	40.03	36.41	38.59	4.08	[36.09, 41.10]
4	44.03	51.30	49.33	47.08	47.94	6.51	[42.98, 52.90]

The measured value of the priority-based sequential task execution strategy for two robots indicates that the average completion time was reduced relative to the single-robot baseline. The variance in completion times (CV 4.08%–7.68%) across all load counts was higher than the single-robot baseline.

C. Thread-Based Concurrent Task Execution

TABLE III
THREAD-BASED CONCURRENT COMPLETION TIME

Loads	Trial 1	Trial 2	Trial 3	Trial 4	Mean (s)	CV (%)	95% CI
2	17.22	14.67	16.87	13.04	15.45	12.71	[12.33, 18.57]
3	25.24	35.38	24.34	26.42	27.85	18.30	[19.74, 35.96]
4	32.22	26.03	35.33	22.87	29.11	19.51	[20.07, 38.15]

Thread-based concurrent execution was the fastest in all load counts. However, the variance in execution times was higher than both the single and priority conditions, and increased with load count (CV 12.71%–19.51%). Thus, while concurrent execution was the fastest among the three conditions, consistency decreased with higher load counts.

D. Comparative Analysis and Discussion

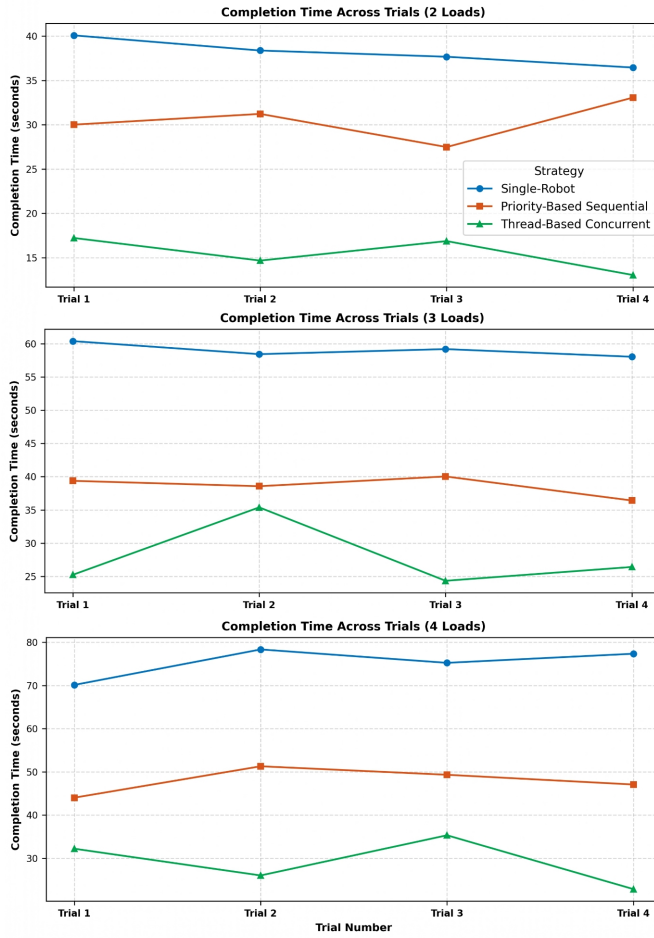


Fig. 7. Comparative Analysis

From Fig. 7, in each of the three conditions, we can see that a single robot has maximum completion times but less variance in those times, and thread-based concurrent execution has minimum completion times but has significant variance in those times whereas priority-based sequential execution sits in between.

TABLE IV
RELATIVE DIFFERENCE IN COMPLETION TIMES AND COEFFICIENTS OF VARIATION

Load	$\bar{T}_P$	$\bar{T}_T$	Improvement	CV_P	CV_T	CV Change
2	30.44	15.45	49.2%	7.68	12.71	39.6%
3	38.59	27.85	27.8%	4.08	18.30	77.7%
4	47.94	29.11	39.3%	6.51	19.51	66.6%

The relative difference in completion time between the two strategies shows that thread-based execution is faster than priority-based execution because each robot moves on to its next command as soon as it finishes the previous one. In contrast, priority-based execution requires both robots to

follow a single loop, causing them to wait for the next cycle before receiving new commands. The relative difference in CV shows that thread-based execution is less predictable because both robots move independently and interfere with each other in different ways in each trial, while priority-based execution is more predictable since Bot 2 checks the distance to Bot 1 and only stops if needed, giving a similar pattern across all trials.

Across all load counts, an independent (unpaired) two-sample t -test with Welch's correction for unequal variances was performed to formally test whether these differences were statistically significant (Table V).

TABLE V
TEST RESULTS (WELCH'S T-TEST)

Loads	T_P (s)	T_T (s)	t	df	p -value	Sig. ($\alpha = 0.05$)
2	30.44	15.45	9.82	5.82	< 0.001	Yes
3	38.59	27.85	4.03	3.57	0.020	Yes
4	47.94	29.11	5.81	4.66	0.003	Yes

The difference in mean completion time between strategies was statistically significant at all three load counts (2 loads: $t(5.82) = 9.82$, $p < .001$; 3 loads: $t(3.57) = 4.03$, $p = .020$; 4 loads: $t(4.66) = 5.81$, $p = .003$), and was further supported by the non-overlapping 95% confidence intervals for priority-based and thread-based completion times at every load count (Tables II–III). The closest margin occurs at three loads, where the priority-based lower bound (36.09 s) sits just above the thread-based upper bound (35.96 s) and the p -value (.020) is correspondingly the weakest of the three conditions, suggesting this is the load count most sensitive to additional trials.

V. CONCLUSION AND FUTURE WORK

This paper compared two task execution strategies: priority-based sequential task execution and thread-based concurrent task execution. Both strategies outperformed a single robot, but with trade-offs. The thread-based approach finished tasks 27–49% faster but had 39–78% more timing unpredictability. The priority-based method was slower but delivered more consistent results. The best strategy therefore depends on whether speed or reliable timing is the priority. A limitation of this study is the small trial count ($n = 4$ per condition); future work should use a larger number of trials per condition to tighten these estimates and strengthen statistical power. In addition, our conclusions are based on a two-robot system operating on a 91*93 cm platform and should not be generalized directly to larger warehouse fleets without further validation. As the robot count increases, coordination overhead will also increase, though characterizing this precisely is left to future work.

Further, this study used a dual-drop-zone setup. A next step is to study a single shared drop zone, where robots must also manage access to a shared resource, and to evaluate additional task execution strategies, such as token-based reservation, to

find a more effective approach for this harder case. We leave this comparison for future work.

REFERENCES

- [1] B. Miloradovic, B. Curuklu, M. Ekstrom, and A. V. Papadopoulos, "Optimizing parallel task execution for multi-agent mission planning," *IEEE Access*, vol. 11, pp. 23815–23830, 2023. [Online]. Available: <https://doi.org/10.1109/ACCESS.2023.3254900>
- [2] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, no. 1, pp. 485–493, 2018. [Online]. Available: <https://doi.org/10.1609/icaps.v28i1.13923>
- [3] P. Ghassemi and S. Chowdhury, "Multi-robot task allocation in disaster response: Addressing dynamic tasks with deadlines and robots with range and payload constraints," *Robotics and Autonomous Systems*, vol. 147, p. 103905, 2022. [Online]. Available: <https://doi.org/10.1016/j.robot.2021.103905>
- [4] T. Kato and R. Kamoshida, "Multi-agent simulation environment for logistics warehouse design based on self-contained agents," *Applied Sciences*, vol. 10, no. 21, p. 7552, 2020. [Online]. Available: <https://doi.org/10.3390/app10217552>
- [5] O. M. Ikumapayi, O. T. Laseinde, R. R. Elewa, T. S. Ogedengbe, and E. T. Akinlabi, "Swarm robotics in a sustainable warehouse automation: Opportunities, challenges and solutions," *E3S Web of Conferences*, vol. 552, p. 01080, 2024. [Online]. Available: <https://doi.org/10.1051/e3sconf/202455201080>
- [6] Z. Qiu, J. Long, Y. Yu, and S. Chen, "Integrated task assignment and path planning for multi-type robots in an intelligent warehouse system," *Transportation Research Part E: Logistics and Transportation Review*, vol. 194, p. 103883, 2025. [Online]. Available: <https://doi.org/10.1016/j.tre.2024.103883>
- [7] S. Fredriksson, Y. Bai, A. Saradagi, and G. Nikolakopoulos, "Multi-agent path finding using conflict-based search and structural-semantic topometric maps," *arXiv preprint arXiv:2501.17661*, 2025. [Online]. Available: <https://doi.org/10.48550/arXiv.2501.17661>
- [8] A. Agrawal, A. S. Bedi, and D. Manocha, "RTAW: An attention inspired reinforcement learning method for multi-robot task allocation in warehouse environments," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2023, pp. 1393–1399. [Online]. Available: <https://doi.org/10.1109/ICRA48891.2023.10161310>
- [9] A. Krnjaic, R. D. Steleac, J. D. Thomas, G. Papoudakis, L. Schäfer, A. W.-K. To, K.-H. Lao, M. Cubuktepe, M. Haley, P. Börsting, and S. V. Albrecht, "Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human co-workers," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2024, pp. 677–684. [Online]. Available: <https://doi.org/10.1109/IROS58592.2024.10802813>
- [10] J. Yin, W. Rao, Y. Xiao, and K. Tang, "Cooperative path planning with asynchronous multiagent reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 5016–5030, 2025. [Online]. Available: <https://doi.org/10.1109/TMC.2025.3526979>
- [11] S. L. Bowman, C. Nowzari, and G. J. Pappas, "Coordination of multi-agent systems via asynchronous cloud communication," *arXiv preprint arXiv:1701.03508*, 2017. [Online]. Available: <https://doi.org/10.48550/arXiv.1701.03508>
- [12] C.-S. Chang, R. Nelson, and D. D. Yao, "Scheduling parallel processors: Structural properties and optimal policies," *Mathematical and Computer Modelling*, vol. 23, nos. 11–12, pp. 93–114, 1996. [Online]. Available: [https://doi.org/10.1016/0895-7177\(96\)00066-0](https://doi.org/10.1016/0895-7177(96)00066-0)
- [13] H. Oates and P. Bercher, "Makespan investigations of sequential, parallel, PO, and POCL plans," in *Proc. AAAI Conf. Artificial Intelligence*, vol. 40, no. 43, pp. 36334–36342, 2026. [Online]. Available: <https://doi.org/10.1609/aaai.v40i43.40953>
- [14] R. Chiva Gil, D. Jarne Ornia, K. A. Mustafa, and J. Alonso Mora, "Predictability awareness for efficient and robust multi-agent coordination," *arXiv preprint arXiv:2411.06223*, 2024. [Online]. Available: <https://doi.org/10.48550/arXiv.2411.06223>