Exploring Planner-Guided Swarms Running on Real Robots

Michael Schader and Sean Luke

George Mason University, Washington, DC 22030, USA
{mschader,sean}@gmu.edu

Abstract. Robot swarms have been proposed as a way to take advantage of the scalability, robustness, and adaptability of natural large-scale multiagent systems in order to solve engineering challenges. However, accomplishing complex tasks while remaining flexible and decentralized has proven elusive. Our prior work on planner-guided robot swarms successfully combined a distributed swarm algorithm implementing low-level behaviors with automated parallel planners and executives selecting high-level actions for the swarm to perform as a whole, but had only been tested in simplistic grid-world simulations. Here we demonstrate our approach on physical robots augmented with experiments in continuous-space simulation, showing that it is an effective and efficient mechanism for achieving difficult task objectives to which swarms are rarely applied.

Keywords: Multi-robot systems and real world robotics, Real-time multi-agent systems, Agent cooperation and negotiation.

1 Introduction

The field of swarm robotics prizes three cardinal virtues. The first virtue is scalability, thanks to potentially large numbers of inexpensive robots. The second is robustness, the ability to withstand the loss of members and to accommodate the addition of new ones. The third is adaptability, the appropriate response to changing conditions in the environment. These virtues take inspiration from natural systems such as ant colonies, flocks of birds, schools of fish, and so on. To achieve these goals, swarm robotics designs have historically taken the form of potentially large numbers of simple and usually homogenous robots, with limited and typically local interaction and communication, and with loosely coupled or entirely separated decision-making.

However, the highly decoupled and distributed nature of a typical robot swarm, valued for these virtues, has also proven difficult to control. As the survey of Brambilla et al. [5] noted, "[d]ue to the lack of a centralized controller, it is in general very difficult to effectively control a swarm once it starts operating." Because they are loosely coupled, swarms by design cannot easily coordinate to do synchronized, interleaved, or nontrivial collaborative tasks. Rather, swarm robotics dogma often turns to *emergent behavior*, arguing that swarms can achieve complex macro-level behavior through the micro-interactions of many agents. But while it is feasible, through simulation, to predict the resulting macrophenomena arising from these interactions, a critical inverse problem — identifying which micro-behaviors will achieve a desired macrophenomenon —

is generally unsolved and perhaps unsolvable. Collective behavior involving synchronization and coordination has proven elusive. In short: researchers have succeeded in getting swarms to forage, patrol, distribute themselves, and to form shapes, but swarms have not shown promise in working together to build a house.

The tension here is between coordination and decentralization. The classic method for identifying, solving, and executing synchronized and collaborative robot tasks is to use a (normally centralized) task planner and executive with tight coupling. But when doing so, a swarm degenerates into a single-agent system with multiple effectors (the swarm robots), hurting scalability due to network complexity, and damaging robustness by relying on a single point of failure. Global knowledge held by every agent would not scale well and would limit adaptability, and requiring long-distance communication among robots would violate the swarm-style focus on having only local interactions.

We are interested in endowing swarm architectures with sophisticated collaboration and synchronization. To this end, we have developed *planner-guided robot swarms* as a novel solution to these problems. In our method, the mission for the swarm is specified in automated planning terms. Each agent has its own planner, and all use the same algorithm and inputs, yielding identical results. The swarm is treated as a set of one or more *virtual agents*, each composed of many real ones and responsible for the parallel execution of the actions in the plan steps. An *a priori* mapping of virtual agent actions to real agent behaviors is the bridge that leads to emergent behavior in service of the mission objectives. Our approach does not require tight synchronization among swarm members but is still robust to retrograde behavior among out-of-sync robots. The method also seeks to ensure that their plans will ultimately synchronize and align.

In our previous work in this area [21, 22], we assumed ideal and simplistic conditions in a trivial simulated grid-world, with predictable communications and none of the sensor noise or action failures associated with actual physical robots. In this paper we remedy this, showing that potentially large groups of physical robots can collectively perform synchronized and planned actions. The robots are able to do these tasks while overcoming physical crowding and interference, significant difficulties in localization and wayfinding, and physical challenges inherent in object detection grasping and manipulation. We further show that the method scales and that it can adapt to dynamic changes in the environment and in the nature of the robot swarm.

We begin with a review of relevant robot swarm research. We then explain our planner-guided method, describing both physical robot and continuous-space simulation implementations. We present the results of experiments performed on real robots as a proof of concept and in simulation as a stress test of time and complexity, showing that our approach lives up to the virtues of scalability, robustness, and adaptability.

2 Previous Work

Planning for Robot Groups Several researchers have worked to use symbolic planning to direct groups of robots. Audrito et al. [2] explored specifying high-level goals using MA-PDDL, the multiagent variant of Planning Domain Definition Language, and automatically translating them into single-agent behaviors using “Aggregate Programming” (AP) to express low-level activities. Jang et al. [10] experimented in simulation with solving underwater survey mission planning challenges with sophisticated versions of PDDL

that included time and numeric variables, soft constraints, and probabilistic actions. Chen et al. [8] used Linear Temporal Logic (LTL), a first-order logic calculus with temporal operators, to define overall objectives, then from that definition synthesized a symbolic plan for robot swarm members to follow. In follow-up work [7], they extended this approach to work in a decentralized fashion. The two-layer mechanism has similarities to our work; however the initial LTL specification and automatically generated plans are constrained to "location and formation-based swarm behaviors", primarily formation control, rather than addressing a general set of possible agent actions. Moarref et al. [15] also used LTL to generate decentralized symbolic plans for individual swarm members to follow, and demonstrated some success in simulation.

Decentralized Swarm Control Much literature has applied or extended traditional decentralized swarm approaches: here we only list a few relevant examples. Bachrach et al. [3] used the "Proto" programming language, designed for directing large groups of agents by treating them like an amorphous medium, to guide swarm robot movements with potential fields. This synthesis of low- and mid-level actions produced robust emergent behaviors, but it did not address high-level goals and decisions. Rossides et al. [20] adapted the particle swarm optimization technique to work with swarms of actual rather than virtual members. They were able to demonstrate using a swam to localize a radio source in physically realistic simulations. The method only applies to swarm movement rather than any other activities. Suárez et al. [23] implemented the "bat algorithm" on real robots, showing that the bio-inspired technique could be used for three-dimensional map-building and navigation. Vardy [25] showed that simple odometry could serve as the basis for swarming behaviors such as aggregation, and in later work [26] developed an algorithm for object clustering also based on simple low-level capabilities.

Physical Swarm Robots Also related to our work is physical demonstrations of noteworthy swarm robot capabilities. Many have explored the classic emergent behavior of foraging along with some variants. Lu et al. [12] and Nouyan et al. [16] implemented collaborative foraging algorithms on both simulated and physical robots. Talamali et al. [24] combined 200 Kilobots (small limited-capability swarm robots) with augmented reality to implement virtual pheromones and to investigate collective foraging. Other researchers have demonstrated different swarm behaviors. Adkikari [1] and Farrugia et al. [9] created physical robot swarms that demonstrated collaborative object transport, the former achieving superlinear performance with a 30-robot team. Chamanbaz et al. [6] implemented a consensus algorithm on physical rovers and floating buoys that allowed the robots to move from one location to another in a coordinated fashion. Petersen et al. [19] created a system for physical and simulated robots to construct pre-planned structures from specialized square blocks.

3 Method

In our planner-guided swarm approach (Figure 1), a human programmer uses the Planning Domain Definition Language (PDDL) [14] to specify a *domain*, the predicates, actions, and objects available in the world; and a *problem*, the initial conditions and goal state of the situation at hand. These definitions are issued to each agent before it enters

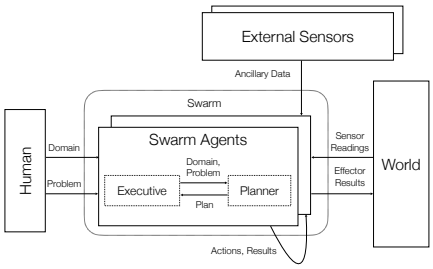


Fig. 1: Planner-guided swarm architecture.

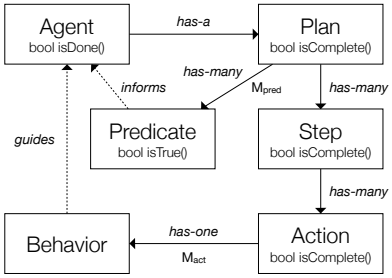


Fig. 2: Component relationships.

the swarm, and are the only centralized element in the architecture. Once online, each agent uses its planner to produce a plan, a series of possibly parallel macro-actions that will accomplish the objectives. If the actions are parallelizable, the agents will distribute themselves (randomly or round-robin, depending on circumstances) to one of several groups, each responsible for one of the concurrent actions. Each agent's executive then begins performing the micro-behaviors associated with its chosen action. As the agents in the swarm interact with the world, collecting sensor inputs and manipulating objects, they exchange information with each other when in close proximity. Optional external sensors, separate from the swarm itself, may provide additional input to agents that are nearby. When the agents determine that a plan step has been completed, they advance to the next one, until the goal has been achieved.

3.1 Software Framework

In our implementation (see Figure 2) the key figure is the *agent*. Each agent uses a built-in planner to generate a *plan*, which encapsulates the result of processing a PDDL domain and a PDDL problem. A plan has one or more *steps*, each of which have one or more *actions*. An action has its own *completion criteria* allowing it to determine when it has been accomplished. A step is complete when all its actions have been accomplished, and a plan is complete when all its steps are complete. Based on the action mapping M_{act}, each action is associated with a single *behavior* which guides the agent to perform certain activities. Last, the predicate mapping M_{pred} associates *predicates* with the plan, one for each predicate specified in the PDDL domain definition, allowing the agent to assess the state of the world in planning terms. When the predicates listed as goal conditions in the PDDL problem statement are all true, the agent is done with the mission.

3.2 Mapping Actions to Behaviors

The action mapping M_{act} ties together the high-level specifications in the PDDL domain and problem statements and the low-level behaviors of each swarm agent. While a classical task planner produces a collection of *actions* that are triggered by initial preconditions, are finite in length, and produce expected postconditions (effects), swarm agent *behaviors* may run forever and do not yield postconditions. Thus to map an action into a behavior, we must define a stopping criterion after which we expect the action

postconditions have been fulfilled by the behavior. Those conditions can be tied to the agent's knowledge of its own activities (such as successfully picking something up), information received from other agents about their activities, data from the agent's sensors, or external data from global sensors helping the swarm.

3.3 Predicate Mappings

The predicate mapping M_{pred} enables replanning by closing the loop between the observed state of the world and the conditions asserted in the problem statement. Although classical planning assumptions do not accommodate changes caused by actors other than the plan-executing agents, rerunning the planner with revised initial conditions provides an effective way to adjust behavior in response to unexpected updates to the environment. The inputs specified in the mapping come from the same categories as those for the action completion criteria: self-knowledge, exchanged information, onboard sensor readings, and external sensor data.

3.4 Algorithm

The general algorithm for each planner-guided swarm agent is as follows:

```
 1: procedure RunDecentralizedAgent(domain, problem)
 2:     currentState ← problem.initialConditions
 3:     while plan is nil or not plan.isComplete do
 4:         if plan is nil then
 5:             plan ← MakePlan(domain, problem)
 6:             stepNum ← 1
 7:             successes ← ∅
 8:         step ← plan.steps[stepNum]
 9:         if step.isComplete then
10:             stepNum ← stepNum + 1
11:             continue
12:         action ← step.actions[groupNum]
13:         if action.isComplete then with probability P_switchGroups
14:             groupNum ← randomly chosen group number
15:             continue
16:         ⟨behavior, parameters, criteria⟩ ← M_act{action}
17:         result ← Execute(behavior, parameters, criteria)
18:         if result = success then
19:             Add new token to successes
20:         TransmitTokens(successes)
21:         for agentSuccesses in ReceiveTokens() do
22:             add tokens in agentSuccesses to successes
23:         newState ← Evaluate(predicates in M_pred)
24:         if currentState ≠ newState then
25:             plan ← nil
26:             problem.initialConditions ← newState
27:             currentState ← newState
28:             continue
```

The functions referenced in this algorithm are:

MakePlan() Run the planner on the supplied domain and problem to generate a new plan, along with any predicates needed to determine its completion status.

Execute() Move as needed and change the world with effectors as per the active behavior and its parameters. This could include activities like driving toward a destination, wandering randomly, picking things up, pressing buttons, assembling structures, and leaving stigmergic messages.

TransmitTokens() Transmit success tokens and state information to neighboring agents. This is how the agents share knowledge of their activities.

ReceiveTokens() Process success token messages received from others. This is the means of coordination that enables the swarm agents to collectively decide when to advance to the next plan step.

Evaluate() Update the state of the world to determine action completion as well as any need for replanning. Each agent evaluates the predicates defined in the planning domain to determine if the environment has changed in a way contrary to the classical assumptions that only a unitary agent can affect the world, thus resyncing the planner to observed reality.

4 Experiments in Simulation and Physical Robot Validation

In this paper we demonstrate our method on physical robots and in a realistic robot swarm simulation, showing its effectiveness and scalability in controlling real groups of robots despite noise, failure, and the complexity of physical constraints.

Fig. 3: Locobot with brick.

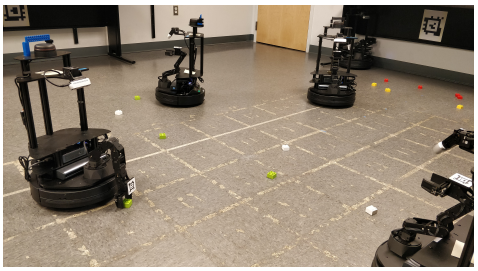
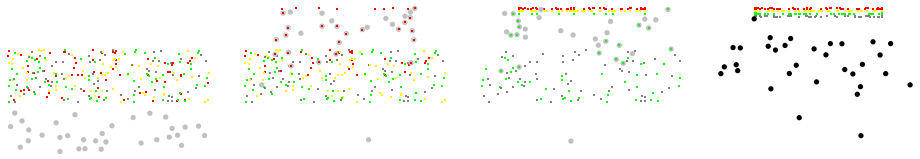


Fig. 4: Robots at work on a Brick Layering scenario. Red and yellow bricks are at their destinations in the top right; green and white ones are still in the field.

We perform three experiments on both physical robots and in simulation. The first is a baseline experiment which demonstrates that the method scales to large numbers of agents (we tested up to 64). The second experiment tests if the method is robust to unexpected changes in the number of agents in the swarm, both tolerating loss of agents and accepting new ones. The third experiment tests if the method can deal with noise and unexpected state changes in the robot environment.



(a) Agents (gray disks) explore the field for bricks (colored rectangles).

(b) Some of the first brick set have reached their destination, others are en route.

(c) The two upper rows are finished; filling in the next row is underway.

(d) All four rows are correctly filled in; the agents have declared the job complete.

Fig. 5: Stages of the Brick Layering scenario in simulation.

Our simulations were created with the MASON multiagent simulation toolkit [13] using two-dimensional continuous-space. For the physical robot implementation we used the Trossen Robotics PX100, a robot platform based on the open-source LoCoBot design (Figure 3). We positioned four AprilTag [17] two-dimensional barcodes on the sides of the experiment area to add visual localization assistance to the onboard odometry, and incorporated YOLOv5 object detection [11] trained on a custom set of images of colored Duplo bricks on the floor.

All code for the abstract swarm operations, including PDDL definitions, parallel planning, success token management, and completion criteria, was shared between the physical robot and simulation implementations. We used our own custom implementation of the GraphPlan algorithm [4] built using the PDDL4J planning toolkit [18].

Base Scenario The experiments we performed are built on a common core challenge we call *Brick Layering* (Figures 4 and 5). The initial state has bricks of four different colors scattered throughout the large field portion of the environment. The goal state has the bricks arranged by color in rows in the target area: red nearest the wall, followed by yellow, then green, and finally white (dark gray in simulation). To achieve the objective, the swarm must move the bricks in the correct sequence, filling in the farthest layers first, or their own movements will disrupt the pattern.

PDDL Definition The planning domain for Brick Layering has one set of predicates to specify if each area is empty, i.e. (area1-empty), (area2-empty), etc; and another to tell what color brick each contains, such as (area1-contains ?item), (area2-contains ?item), and so on, in which ?item is a placeholder for one of the four brick types. The domain also contains actions to fill or empty each area with a particular kind of brick, as in (fill-area1 ?item), (empty-area2 ?item), and the like. The planning problem has initial conditions with all four areas being empty, and a goal state of each area containing the correct color bricks. Driven by these dependencies, the planner produces the following straightforward single-group plan to be executed in sequential order: (fill-area1 red-brick), (fill-area1 yellow-brick), (fill-area1 green-brick), (fill-area1 white-brick).

Action Mapping The crosswalk M_{act} between plan actions and swarm member behaviors is based on translating fill-area and empty-area directives into foraging activities. The

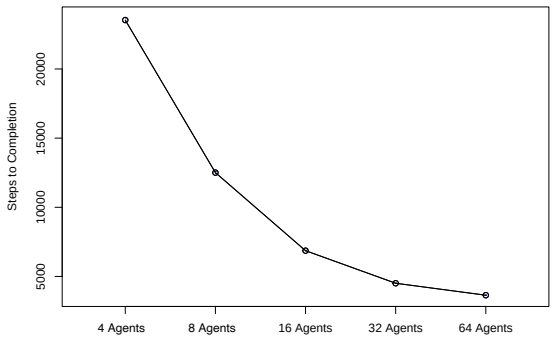


Fig. 6: Scalability in simulation: steps to completion for full runs with various numbers of agents.

Robots	*Minutes to completion*	*Speedup vs one*
1	45.1	1.00
2	24.1	1.87
3	17.6	2.56
4	16.4	2.75

Fig. 7: Scalability results with physical robots.

agents' low-level collect() behavior has parameters for item, source-area, and destination-area. The completion criterion is for the count of fill-area or empty-area events to equal the preconfigured size of each area: four in the physical robot scenario and 64 in simulation. As an example, the plan action (fill-area1 red-brick) maps to the agent behavior collect(red-brick, field, area1), which means, "navigate to the field, then wander looking for a red brick; when you find one, pick it up, navigate to area1, and drop it off; repeat until area1 is full".

Predicate Mapping The mapping M_{pred} between domain predicates and agent knowledge is largely based on the counts of actions accomplished. For example, the truth value of (area4-contains white-brick) is determined by taking the count of (fill-area4 white-brick) events, subtracting the number of (empty-area4 white-brick) events, and seeing if the result equals the size of area4. The only other mechanism in this mapping is used in the third experiment, in which an external sensor determines if red bricks are in area1 or not.

4.1 Experiment 1: Scalability

In the first experiment we examined how the planner-guided swarm method works with different numbers of agents. Ideally, the more robots that are used, the faster the mission will be accomplished, up to the point of negative returns due to crowding and interference among too many individuals. One aspect of this challenge that makes scaling up difficult is contention for the last few bricks of a color, requiring the robots to implicitly or explicitly coordinate their collection actions. Another is the common destination region for robots carrying bricks of the same color, forcing them to navigate around each other on the way to and from each delivery.

In simulation, we set the brick count to 256 and conducted runs with 4, 8, 16, 32, and 64 agents. Average steps to completion ranged from 23,537 with four agents down to 3643 with 64 agents (Figure 6), showing that our method works and scales from a few agents up to common swarm population sizes. We performed 1000 runs of each treatment

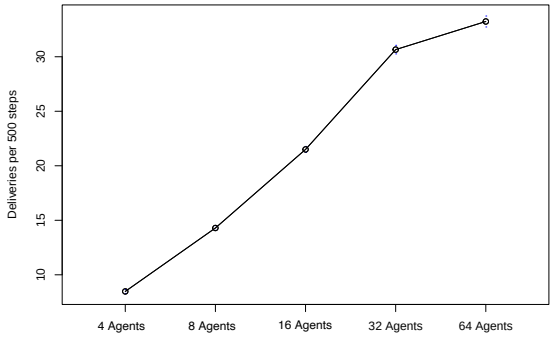


Fig. 8: Robustness in simulation: deliveries per 500 steps for run segments with varying numbers of agents.

Robots	*Deliveries per minute*	*Speedup vs one*
1	0.53	1.00
2	0.73	1.39
3	0.81	1.53
4	1.26	2.39

Fig. 9: Robustness results with physical robots.

in simulation with 100% completion, and verified that all step counts were statistically significant using a two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

For physical validation, we performed full runs of the scenario with sixteen bricks using one, two, three, and four robots, in a much smaller and physically constraining space, so crowding was much more of an issue compared to simulation. We measured the time to completion for each group size and observed that every increase led to faster mission completion (Figure 7). Two robots were 87% faster than one robot, and three robots were 2.5 times as fast as one. With four robots, the speedup increased only to 2.75, showing diminishing returns as expected based on the diameter of the robots and the area of the field. These results showed the scalability of our approach with real robots.

4.2 Experiment 2: Robustness

In the second experiment, we evaluated the planner-guided swarm's ability to continue progressing with a mission (albeit more slowly) when some swarm members are removed, and to speed up progress when new swarm members are added. If working as intended, the swarm will maintain its knowledge of the state of the world regardless of membership changes, tolerating the loss of departing individuals and rapidly incorporating new arrivals. Even the gradual complete replacement of the original swarm members should pose no problem since all are fungible and there is no hierarchy.

In simulation we performed repeated runs with 256 bricks, adding or removing randomly-selected agents every 500 steps to exercise groups with 4, 8, 16, 32, and 64 individuals, while measuring the rate of brick delivery over time (Figure 8). We evaluated 1000 periods of each population size in simulation, and verified that all delivery rates were statistically significant using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction. Periods with more agents were consistently more productive than those with fewer; average rates ranged from 8.5 deliveries per 500-step period up to 33.7. This showed the planner-guided swarm's ability to robustly tolerate removals and take advantage of additions.

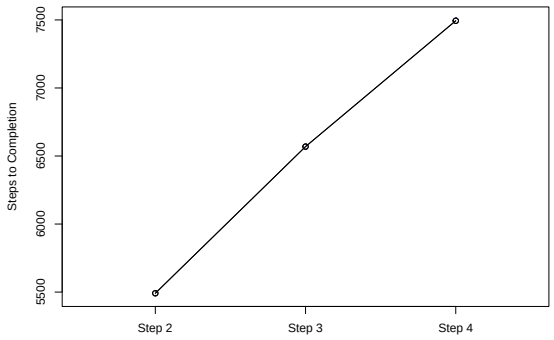


Fig. 10: Adaptability in simulation: steps to completion for 32-agent runs with different change notification times.

Notif. step	*Mins to compl.*	*Add'l mins needed*
2	29.2	5.1
3	31.9	7.8
4	33.7	9.6

Fig. 11: Adaptability results with physical robots.

For physical validation, we launched the sixteen-brick scenario with three robots, then added or removed randomly-selected robots every two minutes to test with 1, 2, 3, and 4 individuals. In spite of this swapping in and out of swarm members, the group continued to progress in the plan, ultimately succeeding. From telemetry we determined the time periods when each number of robots was active and counted the bricks delivered during each period. Delivery rates ranged from 0.53 bricks per minute for one robot up to 1.26 for four (Figure 9), consistent with speedups expected when using more robots.

4.3 Experiment 3: Adaptability

In the third experiment, we explored how a planner-guided swarm can respond to changed circumstances. For this we added a new component to the scenario: a sensor positioned near the brick dropoff area that could observe how many bricks of each color were present. During each run, some time after the first two rows were filled in, we would remove the uppermost layer of red bricks and return them to the field, undoing some of the group's work. After a variable delay, the sensor would inform the swarm members of this change to the world. In response, the individuals would replan based on their updated knowledge of the conditions, remove already-placed bricks as needed to expose the empty top section, then proceed with the usual layering sequence. We would expect the swarm to accommodate the unexpected change and still complete the mission, with earlier notification leading to faster completion than later notification due to the varying amount of rework needed.

In simulation we ran 32 agents through the same series of environmental change scenarios with 256 bricks. For notification after plan steps two, three, and four, the average time steps to completion were 5491, 6569, and 7494 respectively, demonstrating that the sooner the swarm learns about a changed situation, the faster it can replan and ultimately complete the assignment (Figure 10). We performed 1000 runs of each treatment in simulation with 100% completion, and verified that all step counts were statistically significant using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

For physical validation, we used eight bricks and two robots. We started the scenario and allowed the group to complete plan steps one and two, building the red and yellow brick layers. We then removed the first layer and redistributed the red bricks to the field. The sensor noted this change and informed the robots. Upon learning the news, each robot replanned, deciding to disassemble the second layer to gain access to the first, then begin building the four layers again. The revised plans prepended the action (empty-area2 yellow-brick) to the filling steps in the original. Making the same decisions independently, the robots completed the challenge despite the change to their environment.

During successive real robot runs, we delayed notification of this change to the end of plan step three, and then to the end of plan step four. This led to additional preliminary steps being added, i.e. (empty-area3 green-brick) for the step three change and (empty-area4 white-brick) followed by that for the step four change. Across these runs, we observed that the earlier the robots learned about the removal of the top layer, the sooner they could replan and the more quickly they finished the mission (Figure 11).

5 Conclusion

We built on our prior work with planner-guided robot swarms, demonstrating the approach on physical robots and in continuous-space simulation for the first time. We showed that the technique is scalable, robust, and adaptable in real swarm and multi-robot conditions. In future work will build a planner-guided swarm out of larger numbers of physical robots. With it we will experiment with more sophisticated types of collaboration among agents, and evaluate various navigation and communication methods. This will help pave the road from research to real-world applications for this powerful, general approach to swarm engineering.

References

1. Adhikari, S.: Study of Scalability in a Robot Swarm Performance and Demonstration of Superlinear Performance in Conveyor Bucket Brigades and Collaborative Pulling. Ph.D. thesis, The University of Toledo (2021)
2. Audrito, G., Casadei, R., Torta, G.: Fostering resilient execution of multi-agent plans through self-organisation. In: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C). pp. 81–86. IEEE (2021)
3. Bachrach, J., Beal, J., McLurkin, J.: Composable continuous-space programs for robotic swarms. Neural Computing and Applications **19**(6), 825–847 (2010)
4. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial intelligence **90**(1-2), 281–300 (1997)
5. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013)
6. Chamanbaz, M., Mateo, D., Zoss, B.M., Tokić, G., Wilhelm, E., Bouffanais, R., Yue, D.K.: Swarm-enabling technology for multi-robot systems. Frontiers in Robotics and AI **4**, 12 (2017)
7. Chen, J., Sun, R., Kress-Gazit, H.: Distributed control of robotic swarms from reactive high-level specifications. In: 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE). pp. 1247–1254. IEEE (2021)

8. Chen, J., Wang, H., Rubenstein, M., Kress-Gazit, H.: Automatic control synthesis for swarm robots from formation and location-based high-level specifications. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 8027–8034. IEEE (2020)
9. Farrugia, J.L., Fabri, S.G.: Swarm robotics for object transportation. In: 2018 UKACC 12th International Conference on Control (CONTROL). pp. 353–358. IEEE (2018)
10. Jang, J., Do, H., Kim, J.: Mission planning for underwater survey with autonomous marine vehicles. Journal of Ocean Engineering and Technology **36**(1), 41–49 (2022)
11. Jocher, G., et al.: ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements (Oct 2020). https://doi.org/10.5281/zenodo.4154370, https://doi.org/10.5281/zenodo.4154370
12. Lu, Q., Griego, A.D., Fricke, G.M., Moses, M.E.: Comparing physical and simulated performance of a deterministic and a bio-inspired stochastic foraging strategy for robot swarms. In: 2019 International Conference on Robotics and Automation (ICRA). pp. 9285–9291. IEEE (2019)
13. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multiagent simulation environment. Simulation **81**(7), 517–527 (2005)
14. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL: the planning domain definition language (1998)
15. Moarref, S., Kress-Gazit, H.: Decentralized control of robotic swarms from high-level temporal logic specifications. In: 2017 international symposium on multi-robot and multi-agent systems (MRS). pp. 17–23. IEEE (2017)
16. Nouyan, S., Groß, R., Bonani, M., Mondada, F., Dorigo, M.: Teamwork in self-organized robot colonies. IEEE Transactions on Evolutionary Computation **13**(4), 695–711 (2009)
17. Olson, E.: Apriltag: A robust and flexible visual fiducial system. In: 2011 IEEE international conference on robotics and automation. pp. 3400–3407. IEEE (2011)
18. Pellier, D., Fiorino, H.: PDDL4J: a planning domain description library for Java. Journal of Experimental and Theoretical Artificial Intelligence **30**(1), 143–176 (2018)
19. Petersen, K.H., Nagpal, R., Werfel, J.K.: Termes: An autonomous robotic system for three-dimensional collective construction. Robotics: science and systems VII (2011)
20. Rossides, G., Metcalfe, B., Hunter, A.: Particle swarm optimization—an adaptation for the control of robotic swarms. Robotics **10**(2), 58 (2021)
21. Schader, M., Luke, S.: Planner-guided robot swarms. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 224–237. Springer (2020)
22. Schader, M., Luke, S.: Fully decentralized planner-guided robot swarms. In: International Conference on Practical Applications of Agents and Multi-Agent Systems. pp. 241–254. Springer (2021)
23. Suárez, P., Iglesias, A., Gálvez, A.: Make robots be bats: specializing robotic swarms to the bat algorithm. Swarm and Evolutionary Computation **44**, 113–129 (2019)
24. Talamali, M.S., Bose, T., Haire, M., Xu, X., Marshall, J.A., Reina, A.: Sophisticated collective foraging with minimalist agents: a swarm robotics test. Swarm Intelligence **14**(1), 25–56 (2020)
25. Vardy, A.: Aggregation in robot swarms using odometry. Artificial Life and Robotics **21**(4), 443–450 (2016)
26. Vardy, A.: Orbital construction: Swarms of simple robots building enclosures. In: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W). pp. 147–153. IEEE (2018)