Planner-Guided Robot Swarms

Michael Schader and Sean Luke

George Mason University, Fairfax VA 22030, USA
{mschader,sean}@gmu.edu

Abstract. Robot swarms have many virtues for large-scale task execution: this includes redundancy, a high degree of parallel task implementation, and the potential to jointly complete jobs that a single agent could not do. But because of their distributed nature, robot swarms face challenges in large-scale coordination, task serialization or ordering, and synchronization. We investigate the use of a central automated planner to guide a robot swarm to perform complicated, multistep operations normally beyond the capabilities of purely decentralized swarms. The planner orchestrates the actions of task groups of agents, while preserving swarm virtues, and can operate over a variety of swarm communication and coordination modalities. We demonstrate the effectiveness of the technique in simulation with three swarm robotics scenarios.

Keywords: Coordination and control models for multi-agent systems · Knowledge representation and reasoning in robotic systems · Swarm behavior

1 Introduction

Robot swarms have long been difficult to control. In 2004, Gerardo Beni coined the term "swarm robotics" [1] and wrote, "Ultimately, after algorithms for task implementation have been devised, the practical realization requires robustness and this is the result of proper control. Swarm control presents new challenges to robotics engineers." In 2012, Brambilla et al [3] reviewed hundreds of papers in the field and concluded, that "[d]ue to the lack of a centralized controller, it is in general very difficult to effectively control a swarm once it starts operating." Today there is still little ability to specify what a swarm should accomplish and how to have it meet that requirement. A centralized element could solve the control problem by providing a clear point of interface between a human specifying goals and the swarm fulfilling them, and by monitoring and adjusting the swarm's behavior as conditions change. But it is not well understood how introducing a limited amount of central control would affect those virtues that flow from decentralization.

We explore the notion of marrying a distributed behavior-based swarm with a centralized planner. Swarms permit parallelism, redundancy, and simple agent definitions, while central planning provides coordination for nontrivial tasks involving sequencing and heterogeneous behavior which are very difficult for swarms to do on their own.

This work sits at two different intersections within research and practice. First, in autonomous robotics, there is a long-standing tension between *deliberative* approaches, in which agents choose actions based on a model of the world around them, and *reactive* methods, in which sensor input is closely coupled to behavior without extraneous intermediate layers. A *planner-guided swarm*, described here, is a hybrid architecture combining planning (model-driven deliberation) with reactivity (robots with only local knowledge following simple rules) to get the best of both worlds.

Second, in multi-robot systems, there is a dichotomy between centralized and decentralized architectures. Many multi-robot implementations in industry use a fully centralized approach, with each robot directed in real time by a master controller. Swarm robotics, on the other hand, focuses on independent agents executing simple behaviors that add up to emergent results. A planner-guided swarm preserves the flexibility and robustness of a swarm while adding a bit of centralized direction in order to achieve far more complicated results.

In this paper we first survey the work done by other researchers on the challenge of swarm control and show that no one has previously explored the addition of a central planner to a decentralized swarm. Next, we describe the overall architecture and individual components of our planner-guided swarm concept. We explain how this architecture works with different communication modalities and is independent of the internal design of the agents. Finally, we lay out three very different scenarios that we used to test the capabilities of the system in simulation, and explain how the experiments showed the value of our approach.

2 Previous Work

The most common method seen in the literature to program robot swarms is carefully crafting finite state machines that run on each agent. For example, [10] used regular languages to define Finite-State Automata (FSA) dictating agent behavior. [12] developed a Probabilistic Finite State Machine (PFSM) that used a potential field to drive swarm robot search behavior. [13] progammed swarms by generating behavior trees.

Considering fully decentralized methods, [16] pushed pure pheromone swarm robotics to the limit by implementing the five tasks of classic compass-straightedge geometry, suggesting an upper bound on what such a swarm can accomplish reasonably efficiently without the addition of at least some global knowledge and interaction. [18] applied Embodied Evolutionary Robotics (EER) methods to the development of swarm agent behavior that leads to desired emergent outcomes; tasks were limited to navigation and foraging, and centralized planning was not considered.

Combining reactive and centralized ideas, [21] created a swarm control mechanism in which a subset of the robots was given global information and special influence over the others, occupying a midpoint between centralized and decentralized approaches. [4] developed a decentralized framework for Multi-Agent Systems planning using Hierarchical Task Networks (HTNs) but not emphasizing swarm concepts.

Others have focused on higher-level issues in lightly-centralized swarm design. [6] examined the tradeoffs between microscopic (robot-level) and macroscopic (swarm-level) design and engineering, proposing a hybrid solution. [8] built a top-down systems engineering approach to planning and operating swarms of UAVs.

In the area of multi-agent coordination, [5] created a formal protocol for swarm robots to exchange information about stigmergic values and activities, showing a potential building block for a planning system to interface with pheromone swarms. [20] presented multiple biologically-inspired algorithms for coordinated motion on real robots using pheromone-like signals to influence their behavior. [9] explored the use of response thresholds in determining when agents in a swarm should change behavior.

Task allocation has been examined from several perspectives. [22] proposed the use of a market-based mechanism for swarm robots to acquire task assignments, in which self-interested agents participate in a virtual economy. [2] implemented virtual pheromones in robots built on the Belief-Desire-Intention framework (BDI). [14] used the Artificial Bee Colony (ABC) algorithm to assign tasks in a multi-agent system, achieving results comparable to other methods but with greater efficiency.

[19] developed Termes, a swarm construction system. A compiler takes an architectural specification (with some important constraints) as input and generates a specialized plan for a swarm of robots to build it. They simplify as many aspects of the environment as possible, using custom blocks that are placed in an iterative fashion. Once the plan is loaded onto the robots, all interaction is stigmergic (by where blocks have been placed).

[7] developed AntLab, a centralized multi-robot system which decomposes problems and directs varying numbers of robots to solve them. It is flexible in terms of what tasks it performs and how it handles adding or removing robots from its set of workers. However, it is not a swarm: the individual robots have their motions and activities planned in advance by a central controller, and there is no communication among the robots.

3 Method

It is clear that classic swarm mechanisms can be used to accomplish certain tasks in the real world. However the tasks explored in previous work, such as coverage, foraging, and navigation, are generally simple and known to be well suited to decentralized solutions. It is not clear how to elicit sophisticated behavior from a swarm in situations that require coordinated, simultaneous activity.

The gap we identified in the literature is the lack of research into hybrid architectures that separately address high-level and low-level behavior. Previous work explores methods to design and deploy swarm agent behaviors intended to achieve certain goals, but without the ability to monitor and manage the swarm's actions at runtime. Other work features explicit central control of each individual agent in order to optimize its activity, but sacrifices swarm characteristics such as redundancy and local-only communication in the process. Our innovation is to split the swarm design work into high-level planning and low-level behavior implementation, map planning actions to specific behaviors, and build an architecture that manages emergent behavior to achieve desired ends.

The collection of agents used in our approach is truly a swarm, and we focus on this subset of multiagent systems. The individuals are homogeneous, communicate only locally, have a small number of simple behaviors, and exhibit emergent behavior that advances the system toward a goal. They are guided by an external component that adjusts behavioral parameters for groups of agents within the swarm. This guidance is driven by the output of an automated planner. The plan domain and problem definitions are written by a human controller, and the agent behaviors and the mapping from plan actions to agent activity are developed for the situation at hand; all other aspects (agent

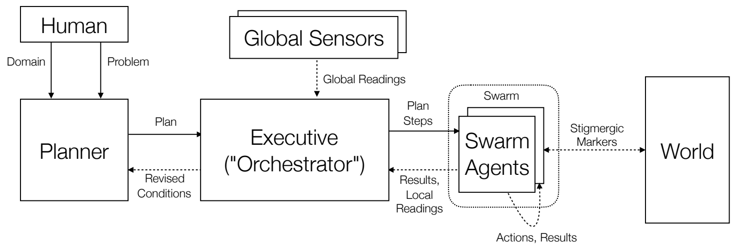


Fig. 1: Planner-guided robot swarm architecture

internals, communication modality, monitoring mechanisms) are generalized and can be used in any scenario.

The architecture of our approach is shown in Figure 1. The process begins with a *human* creating a domain definition that describes how the world works in classical planning terms, and a problem definition that lays out the initial and goal states. These definitions are given to the *planner*, which devises a sequence of (potentially parallel) actions that will achieve the desired result. The planner submits the plan to the executive component, called the *orchestrator*, which maps the actions in the plan to specific behaviors for the agents to perform. The orchestrator divides the *swarm agents* into task groups, and then transmits the first set of instructions to them. The agents operate according to the parameters they have been given, with overall progress emerging from their activity. The agents may employ stigmergy (communication via changes made to the *world*) and other swarm-type mechanisms to transmit information, such as pheromones, direct interaction, and short-range communication. The orchestrator monitors the swarm's progress through any combination of data returned by the agents and *global sensors* that track conditions in the world. When the orchestrator determines that the current plan step has been completed, it issues the next set of instructions, advancing until the plan has been completed and the goal state achieved.

3.1 Planner

A human creates Planning Domain Definition Language (PDDL) files that specify the domain (the predicates available to evaluate the world and the actions the agent can take to change it) and the problem (the initial and goal states) [17]. These files are fed into the planner, which generates a plan in the form of a sequence of actions that will change the world from the initial state to the goal state. These plans may be sequential or parallel. This capability is exposed as the function MakePlan(*domain, problem*).

In our experiments we used the Blackbox planner [11]. It is well suited to our purposes in that it accepts and emits standard PDDL, generates parallel plans with any number of simultaneous actions, is available for each major operating system, and is highly performant. Nothing in our method is dependent on the particulars of Blackbox; any PDDL-based planner will do.

3.2 Orchestrator

The orchestrator is the interface to the multi-agent swarm. To draw a comparison with the classic three-tiered agent architecture, the orchestrator is the Executive, ensuring that the output of the Deliberator (the planner) is executed and monitored properly. This component transforms plans into sequenced actions that the agents can perform, loosely oversees progress without explicitly conducting the individuals in the swarm, and potentially makes plan and assignment adjustments if tasks are not being accomplished.

Agent Setup The orchestrator divides the set of agents into however many task groups are called for in the scenario. This size of each group may be the same based on splitting up the total number of agents, or may vary depending on how many are needed for each task. The orchestrator tells each agent which task group it is in, and from then onward, the agents know which group instructions to follow. In some situations it is useful to have a task group that is not given any instructions, but rather serves as a communications medium by having its agents wander around exchanging information with others that are performing specific tasks.

Action to Behavior Translation The mapping of plan actions to agent behaviors is determined at design time for the scenario. For each action listed in the domain definition, the developer implements code that determines what parameters need to be given to the agents to have them execute the appropriate behavior. For example, an action to open a door might translate to the agent behavior of traveling to the correct button and pressing it, while an action to build a wall could become a set of parameters instructing the agents to wander looking for wall material and bringing it to a designated location.

Progress Monitoring After the orchestrator issues actions to the task groups, it monitors for completion of the plan step. One way it can do this monitoring is via the agents, counting the number reporting success or tracking other observations that the agents bring back. An alternative method is to use global sensors: mechanisms that allow the orchestrator to directly assess the state of the world in order to know when success has been achieved. Upon learning of step completion, either through diffused knowledge carried by the agents or through direct observation, it moves onto the next step. Ultimately, the orchestrator recognizes when the goal state has been attained.

Regardless of the problem definition or the communication modality, the core algorithm executed by the orchestrator is the same. While there is a step in the plan remaining, we iterate as follows. First, for each action in the step, we convert this action into a concrete agent behavior. Then the task is assigned to an available task group, and the action completion criteria are added to *conditions*. Second, we wait until every condition in *conditions* has been met.

3.3 Agents

The agents are the members of the swarm. They perform actions in the world based on their programming and the guidance they receive from the orchestrator. When accomplishing the objective requires collaboration among the agents, it may be top-down, in which the planner instructs different groups to take actions that complement each other, e.g. one pushes a button to open a door while another goes through it. Alternatively, it may be bottom-up, in which the agents themselves interact to perform a function, such as joining up to move an object. (See Section 4.3 for our experiment that included both.)

In our conceptual model, there are no restrictions on the internals of the swarm agents. The only requirement is for them to receive behavior parameter information from the orchestrator and adjust their activity accordingly. Our implementation uses state machines to transition among various behaviors, with variables (such as destinations and object types) that are set based on the orchestrator's messages. Note that the orchestrator has no knowledge of these internal states. Agents can also be built using a subsumption architecture, or with complex deliberation, or with any other mechanism that allows them to act on guidance from the orchestrator.

For each scenario (or real-world situation), we craft a planning description of the domain (e.g. locations, objects, constraints) as well as the low-level behaviors of the agents (such as move, pick up, put down). We then create a mapping of domain actions to agent behaviors ("move block A to location 2" becomes "search for an A-type object, pick it up, navigate to location 2, put it down"). Next we implement any needed success criteria checks ("Is location 2 full of A-type objects?"). Finally, we specify the problem in planning domain terms and send the swarm to do the job.

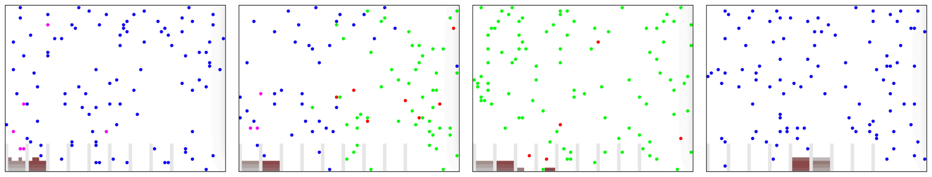
3.4 Communication modalities

The planner-guided approach depends on having some means of communication between the orchestrator and the agents: the agents need to receive guidance based on the plan, and they need to transmit status updates back to the orchestrator. There are several different communication modalities typically used by swarms, so we have accommodated all of them in our architecture and experiments (see Section 4.1):

1. *Broadcast*: The agents instantly learn the particulars of the action they need to execute, and transmit their results directly back to the home base. This method is fast and effective, but it depends on a permissive communications environment and a one-to-many architecture with limited scalability.
2. *Direct Interaction*: Messages diffuse through the swarm by being exchanged when individuals make physical contact with each other or the home base. A variant of this is *Local Interaction*, in which messages are also sent peer-to-peer, but with a relaxed proximity requirement (nearby rather than touching).
3. *Pheromones*: Information is embedded in the environment and sensed when agents pass over it; this removes the need to be in the same place at the same time.

4 Experiments

We have developed three different scenarios in the course of this work, each emphasizing different multiagent control challenges. Our intent is to show that the planner-guided swarm approach is effective and scalable over a variety of situations, and with several different communication mechanisms. (Future work will address other challenges such as responding to unexpected events). The first scenario, *Blocks World*, demonstrates



(a) The Swarm starts rearranging blocks so the top ones end up on the bottom

(b) The agents complete Step 1, word gets back, and the next action spreads

(c) All the agents have moved onto Step 2 and are moving the blocks as directed

(d) Ultimately, the blocks are rearranged as specified and the task is completed

The blue and green dots represent agents on even- or odd-numbered steps. The gray/brown shaded pieces make up the eight different blocks. The thin vertical lines are walls.

Fig. 2: Stages of the Blocks World scenario

complex sequencing; the second, *MarsOne*, shows a more realistic (albeit on Mars) exploration and construction environment; and the third, *Airlocks*, is a locked-room situation requiring both micro- and macro-level coordination and sequencing.

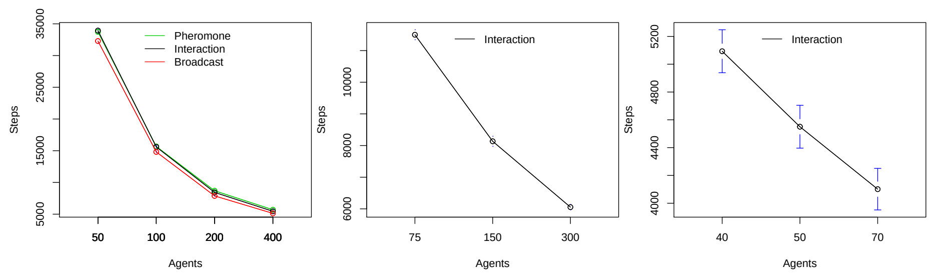
Each scenario is defined by a domain definition and problem definition formalized in PDDL and shown in the Appendix (Section 6). We ran all the simulations using the MASON multiagent simulation toolkit [15], collecting the number of steps needed to succeed under each treatment.

4.1 Blocks World: Complex parallel manipulation in idealized space

The first scenario, Blocks World, provides us with a well-understood proving ground for testing the integration of classical planning with a robotic swarm. The actions need to be performed in a correct complicated sequence in order to succeed. Some steps of the plan allow actions to be performed in parallel, offering a valuable speedup in completion time. However, other steps have dependencies which don't permit full parallelization, and the swarm needs to handle this effectively. This challenge showed that we could perform interleaved parallel and serial tasks with a swarm.

Blocks World is derived from the classic planning problem using labeled blocks that can be placed on a table or on each other (Figure 2). In the swarm simulation, each block consists of several objects that can each be moved by an agent. Walls separate the block sources and destinations in order to force ordered planning to rearrange them efficiently. The domain definition allows up to four actions to take place concurrently, equivalent to having four hands grabbing, holding, and placing blocks in the environment. The initial state features two stacks of four blocks each (ABCD and EFGH, from bottom to top). The goal is to have two stacks in the same order, except with the bottom block of each moved to the top position (BCDA and FGHE).

In this scenario, we use four task groups, correlating conceptually to four manipulator arms that can move blocks simultaneously. The orchestrator translates each of the actions defined in the domain (PICK-UP, PUT-DOWN, UNSTACK, and STACK) to an agent directive. PICK-UP and UNSTACK are mapped to foraging-type behavior in which the agent explored the area looking for the specified type of block. PUT-DOWN and STACK



(a) Blocks World (b) MarsOne (c) Airlocks

Fig. 3: Mean steps to completion of scenario

are mapped to the inverse: seeking a destination and depositing the right type of block there. The completion criteria are based on counting the number of blocks picked up or deposited correctly for each action.

The agents are controlled by two parameters: which kind of block to seek, and where to place the type of block they were holding. For example, "PICK-UP H" translates to "find an item of type H and pick it up", while "STACK H G" becomes "if holding an item of type H, navigate to the site above the G blocks and drop it off". The agents have two states, Exploring and Carrying, and two parameters, what to find and where to put it. While Exploring, they use whatever means are at their disposal to find the specified item. Once they find it, they switch to Carrying, and work to bring their carried objects to the destination. Upon reaching it, they drop off their items and resume Exploring. The agents navigate collaboratively using pheromone gradients; each lays down a trail upon leaving a site in order to help others find the same location.

We tested this scenario using three different communication mechanisms explained in Section 3.4: Broadcast, Direct Interaction, and Pheromone Encoding. For each modality, we varied the number of agents from 50 up to 400 to observe the effect on the average number of steps needed to reach the goal state (Figure 3a). We performed 1000 runs of each treatment and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

As the Blocks World scenario with various communication mechanisms was run with increasing numbers of agents, swarm performance improved until leveling off. This was due to interference caused by having too many agents in a contained space. The strong similarity of the curves for the three different information exchange modalities indicates that performance is hardly affected by the means of transmission. This implies that the planner-guided approach to swarm control is flexible and not dependent on any particular communication architecture.

4.2 MarsOne: Simulated autonomous construction with dependencies

Compared to the first scenario, the second provides a more physically realistic environment with more meaningful objectives to accomplish. MarsOne is based on plans to build a Martian colony in advance of human explorers by using autonomous robots (Figure 4). In the MarsOne domain, a ship contains equipment components for an antenna,

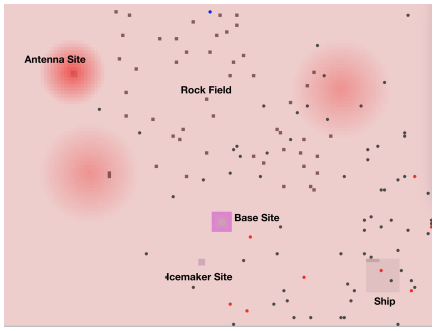


Fig. 4: Initial state of the MarsOne scenario

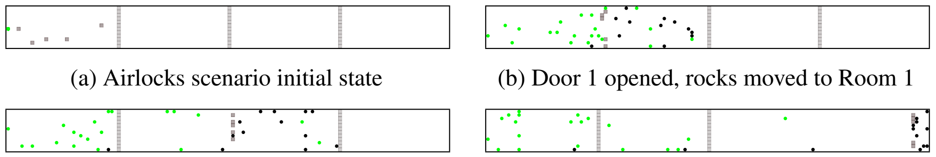
a machine that can produce ice blocks, and the modules for the base. Useful rocks are strewn about the area, and a distant hill provides an ideal radio transmission point. As per the problem definition, the planner-guided swarm is required to find the hill, bring the antenna components to it and assemble them, emplace the base modules, build the ice-making machine, collect the ice blocks it produces and use them to build a wall around the base, and finally gather rocks and put them around the ice wall.

The actions and success criteria are more sophisticated and open-ended than in Blocks World. For example, ice blocks can only be collected as they are produced, one at a time; ice and rock must be placed into concentric rings in the right sequence. This greater complexity only applies to the planner; the agents and the mapping to their actions are as simple as those in Blocks World, with COLLECT and DEPOSIT-AT instructions working like PICK-UP and PUT-DOWN in the previous scenario. Each action specifies a type of object and a source or destination, e.g. “COLLECT base-parts ship” means “find the ship and pick up a base part”, and “DEPOSIT-AT base-parts base” means “if carrying a base part, find the base and put it down there”. DEPOSIT-AT-SURROUNDING works exactly like DEPOSIT-AT from the agents’ perspective; the distinction between the two is needed to force proper sequencing by the planner (for the ice wall to surround the base and the rock wall to surround the ice wall, the ice wall must be built first).

We tested this scenario using the Direct Interaction communications mechanism and varied the number of agents from 75 up to 300 to observe the effect on the average number of steps needed to reach the goal state (Figure 3b). We performed 1000 runs of each treatment and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction. This challenge showed whether the planner-guided swarm approach could handle believable, less-idealized situations that were very different from the more abstract world in the first scenario. Under testing, the swarm was able to execute the planned steps and reach the goal state.

4.3 Airlocks: Mandatory coordination among partitioned task groups

The third scenario, Airlocks, has a sequence of rooms behind locked doors (Figure 5). The agents needs to move rocks from the Room 0 starting area (on the far left), through Room 1 and Room 2, into Room 3 (on the far right). These rocks are heavy and can only


(a) Airlocks scenario initial state

(b) Door 1 opened, rocks moved to Room 1

(c) Door 1 closed, Door 2 opened, rocks moved to Room 2

(d) Door 2 closed, Door 3 opened, rocks moved to Room 3

Fig. 5: Stages of the Airlocks scenario

be moved by pairs of agents working in unison. Each of the doors has a button that opens it as long as an agent is holding the button. Once the button is released, that door closes. Finally, only one of the three doors can be open at a time. Note that the agents in this scenario can not rely on physical contact to transfer information, because closed doors would partition the swarm into separate regions. For this reason, they have a short-range transmission capability (a 3-square radius) that allows them to communicate through closed doors. The domain definition captures the mutually exclusive relationship among the door states; the problem definition simply states that the world begins with the doors closed and the rocks in Room 0, and should end with the rocks in Room 3.

We tested this scenario using Direct Interaction and varied the number of agents from 40 up to 70 to observe the effect on the average number of steps needed to reach the goal state (Figure 3c). We performed 1000 runs of each treatment and verified for statistical significance using the two-tailed t-test at $p = 0.05$ with the Bonferroni correction.

The Airlocks scenario requires coordination among the agents at two different levels. From the whole-swarm perspective, the opening and closing of each door has to be synchronized with the movement of the agents through it, and with the state of the other doors. From the individual perspective, agents need to coordinate with each other to collaboratively transport the heavy rocks (using a grab-and-hold approach). This challenge examined the planner-guided swarm's ability to operate with mandatory coordination requirements while executing a centrally-developed plan. Testing showed that the swarm could solve this problem while scaling up successfully.

5 Conclusions and Future Work

The planner-guided swarms were highly effective at solving all three scenarios. These different situations were handled with no change to the core algorithm of the orchestrator, and only minor practical adjustments to the low-level agent behaviors. The swarm performed similarly well when tested with various communications modalities, showing that the concept is independent of any specific information exchange mechanism. These experiments demonstrated the generality of the approach and its applicability to a wide variety of environments.

As future work, we will explore solving the problem of retrograde behavior (some agents being stuck on a previous plan step); ensure swarm robustness via task reallocation, adjusting communications strategies, and other methods; and demonstrate that coordination under planning can be performed to accomplish tasks reliably and efficiently. This work will show for the first time a flexible, reusable solution to the inverse problem of controlling emergent swarm behavior without sacrificing robustness or low-level decentralization.

References

1. Beni, G.: From swarm intelligence to swarm robotics. In: International Workshop on Swarm Robotics. pp. 1–9. Springer (2004)
2. Bottone, M., Palumbo, F., Primiero, G., Raimondi, F., Stocker, R.: Implementing virtual pheromones in bdi robots using mqtt and jason (short paper). In: Cloud Networking (Cloudnet), 2016 5th IEEE International Conference on. pp. 196–199. IEEE (2016)
3. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence **7**(1), 1–41 (2013)
4. Cardoso, R.C.: A decentralised online multi-agent planning framework for multi-agent systems (2018)
5. Chibaya, C.: An xset based protocol for coordinating the behaviour of stigmergic ant-like robotic devices. In: Proceedings of the 2015 Annual Research Conference on South African Institute of Computer Scientists and Information Technologists. p. 9. ACM (2015)
6. Durand, J.G.D.S.: A methodology to achieve microscopic/macroscopic configuration tradeoffs in cooperative multi-robot systems design. Ph.D. thesis, Georgia Institute of Technology (2017)
7. Gavran, I., Majumdar, R., Saha, I.: Antlab: a multi-robot task server. ACM Transactions on Embedded Computing Systems (TECS) **16**(5s), 190 (2017)
8. Giles, K.: Mission based architecture for swarm composability. Tech. rep., Naval Postgraduate School Monterey United States (2018)
9. Kanakia, A.P.: Response threshold based task allocation in multi-agent systems performing concurrent benefit tasks with limited information (2015)
10. Kaszubowski Lopes, Y.: Supervisory Control Theory for Controlling Swarm Robotics Systems. Ph.D. thesis, University of Sheffield (2016)
11. Kautz, H., Selman, B.: Blackbox: A new approach to the application of theorem proving to problem solving. In: AIPS98 Workshop on Planning as Combinatorial Search. vol. 58260, pp. 58–60 (1998)
12. Khan, M.S., HASAN, M., Ahmed, T.: A new multi-robot search algorithm using probabilistic finite state machine and lennard-jones potential function (2018)
13. Kuckling, J., Ligot, A., Bozhinoski, D., Birattari, M.: Behavior trees as a control architecture in the automatic modular design of robot swarms. In: International Conference on Swarm Intelligence. pp. 30–43. Springer (2018)
14. Liu, H., Zhang, P., Hu, B., Moore, P.: A novel approach to task assignment in a cooperative multi-agent design system. Applied Intelligence **43**(1), 162–175 (2015)
15. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: Mason: A new multi-agent simulation toolkit. In: Proceedings of the 2004 swarmfest workshop. vol. 8, pp. 316–327. Michigan, USA (2004)
16. Luke, S., Russell, K., Hoyle, B.: Ant geometers. In: Proceedings of the European Conference on Artificial Life 13. pp. 100–107. MIT Press (2016)
17. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: Pddl-the planning domain definition language (1998)
18. Pérez, I.F.: Distributed Embodied Evolutionary Adaptation of Behaviors in Swarms of Robotic Agents. Ph.D. thesis, Université de Lorraine (2017)
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. Shirazi, A.R.: Bio-Inspired Self-Organizing Swarm Robotics. Ph.D. thesis, University of Surrey (United Kingdom) (2017)
21. Trabattoni, M., Valentini, G., Dorigo, M.: Hybrid control of swarms for resource selection. In: International Conference on Swarm Intelligence. pp. 57–70. Springer (2018)
22. Yusuf, F.: Multi robot task allocation using market based approach. Ph.D. thesis, Universiti Tun Hussein Onn Malaysia (2015)

6 Appendix: PDDL Files

Blocks World domain and problem definitions

```
(define (domain BLOCKS-WORLD) (:requirements :strips :typing) (:types block)
  ;; Actions ending with "N" are expanded by a preprocesor into "pick-up1", "pick-up2", etc.
  (:predicates (on ?x - block ?y - block) (ontable ?x - block) (clear ?x - block) (handempty) (holding ?x - block))
  (:action pick-upN :parameters (?x - block) :precondition (and (clear ?x) (ontable ?x) (handempty))
    :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))
  (:action put-downN :parameters (?x - block) :precondition (holding ?x)
    :effect (and (not (holding ?x)) (handempty) (clear ?x) (ontable ?x)))
  (:action unstackN :parameters (?x - block ?y - block) :precondition (and (on ?x ?y) (clear ?x) (handempty))
    :effect (and (holding ?x) (not (handempty)) (clear ?y) (not (clear ?x)) (not (on ?x ?y))))
  (:action stackN :parameters (?x - block ?y - block) :precondition (and (holding ?x) (clear ?y))
    :effect (and (not (holding ?x)) (handempty) (not (clear ?y)) (clear ?x) (on ?x ?y))))

(define (problem REORDER) (:domain BLOCKS-WORLD) (:objects a b c d e f g h - block)
  (:init (handempty) (clear d) (clear h) (ontable a) (ontable e) (on d c) (on h g) (on c b) (on g f) (on b a) (on f e))
  (:goal (and (on a d) (on e h) (on d c) (on h g) (on c b) (on g f))))
```

MarsOne domain and problem definitions

```
(define (domain MARS-ONE) (:requirements :strips :typing) (:types group thing site)
  (:predicates (at ?t - thing ?s - site) (reachable ?s - site) (carrying ?g - group ?t - thing)
            (surrounds ?t - thing ?s - site) (empty ?g - group) (discoverable ?s - site)
            (produces ?s - site ?parts ?output - thing) (encloses ?outsite ?insite - site))
  (:action collect :parameters (?g - group ?t - thing ?s - site) :precondition (and (empty ?g) (reachable ?s) (at ?t ?s))
    :effect (and (not (empty ?g)) (carrying ?g ?t) (not (at ?t ?s))))
  (:action deposit-at :parameters (?g - group ?t - thing ?s - site) :precondition (and (carrying ?g ?t) (reachable ?s))
    :effect (and (empty ?g) (not (carrying ?g ?t)) (at ?t ?s)))
  (:action deposit-at-surrounding :parameters (?g - group ?t - thing ?outsite ?insite - site)
    :precondition (and (carrying ?g ?t) (reachable ?insite) (reachable ?outsite) (encloses ?outsite ?insite))
    :effect (and (empty ?g) (not (carrying ?g ?t)) (surrounds ?t ?insite) (not (reachable ?insite)) (at ?t ?outsite)))
  (:action activate :parameters (?s - site ?parts ?output - thing)
    :precondition (and (produces ?s ?parts ?output) (at ?parts ?s)) :effect (and (at ?output ?s)))
  (:action discover :parameters (?g - group ?s - site) :precondition (and (empty ?g) (discoverable ?s)) :effect (and (reachable ?s))))

(define (problem BUILD-BASE) (:domain MARS-ONE) ; References to "groupN" are expanded into "group1", "group2," etc.
  (:objects groupN - group rocks - thing scattered - site ship antenna icemaker base
          ice-wall rock-wall antenna-parts icemaker-parts base-parts ice-blocks)
  (:init (empty groupN) (produces icemaker icemaker-parts ice-blocks) (reachable ship) (reachable icemaker)
        (reachable scattered) (reachable rock-wall) (reachable base) (reachable ice-wall) (at antenna-parts ship)
        (at icemaker-parts ship) (at base-parts ship) (at rocks scattered) (discoverable antenna)
        (encloses ice-wall base) (encloses rock-wall ice-wall))
  (:goal (and (at antenna-parts antenna) (at base-parts base) (surrounds ice-blocks base) (surrounds rocks ice-wall))))
```

Airlocks domain and problem definitions

```
(define (domain AIRLOCKS) (:requirements :strips :typing)
  (:predicates (opened-r1) (opened-r2) (opened-r3) (closed-r1) (closed-r2) (closed-r3) (in-r0) (in-r1) (in-r2) (in-r3))
  (:action open-r1 :precondition (and (closed-r1) (closed-r2) (closed-r3)) :effect (and (opened-r1) (not (closed-r1))))
  (:action open-r2 :precondition (and (closed-r1) (closed-r2) (closed-r3)) :effect (and (opened-r2) (not (closed-r2))))
  (:action open-r3 :precondition (and (closed-r1) (closed-r2) (closed-r3)) :effect (and (opened-r3) (not (closed-r3))))
  (:action close-r1 :precondition (opened-r1) :effect (and (not (opened-r1)) (closed-r1)))
  (:action close-r2 :precondition (opened-r2) :effect (and (not (opened-r2)) (closed-r2)))
  (:action close-r3 :precondition (opened-r3) :effect (and (not (opened-r3)) (closed-r3)))
  (:action move-to-r1 :precondition (and (in-r0) (opened-r1)) :effect (and (not (in-r0)) (in-r1)))
  (:action move-to-r2 :precondition (and (in-r1) (opened-r2)) :effect (and (not (in-r1)) (in-r2)))
  (:action move-to-r3 :precondition (and (in-r2) (opened-r3)) :effect (and (not (in-r2)) (in-r3))))

(define (problem MOVE-ROCKS) (:domain AIRLOCKS) (:init (in-r0) (closed-r1) (closed-r2) (closed-r3)) (:goal (in-r3)))
```