Multiple Robots For Multiple Missions: Architecture for Complex Collaboration

Noa Agmon[1], **Oleg Maximov**[1], **Ariel Rosenfeld**[2], **Shai Shlomai**[1], **Sarit Kraus**[1]
[1]Department of Computer Science, Bar-Ilan University, Israel
[2]Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Israel
agmon@cs.biu.ac.il, oleg@maksimov.co.il, arielros1@gmail.com, shaishlomai@gmail.com, sarit@cs.biu.ac.il

Abstract

As systems of multiple robots become more prevalent both in research and in real world applications, they may be required to handle new missions, collaborate in teams they are not necessarily familiar with, and react, during mission execution, to changes in both the environment and in their own abilities. In this paper we describe a six-layered architecture that handles inherently these challenges, as well as allowing human operator to oversee and intervene whenever needed. We have implemented the architecture and demonstrate the activity flow of execution, namely: how does the system and its components react to dynamic changes. We have conducted experiments in both simulation and in real robots, and show the efficiency of the system in two different parameters: cost of coordination and communication, and mission performance. The implementation is open source, and available for future use.

1 Introduction

Multi-Robot systems have been successfully deployed in different real-world tasks such as fire-fighting [Saez-Pons *et al.*, 2010], landmine detection [Dasgupta *et al.*, 2015], decontamination of radiation [Kaiman, 2016], agricultural work [Auat Cheein and Carelli, 2013], construction [Ardiny *et al.*, 2015], underwater missions [Kulkarni and Pompili, 2010], warehouse operation [Guizzo, 2008] and Search And Rescue (SAR) [Liu and Nejat, 2013]. A common theme amongst these systems is that they are designed to accommodate a *single* mission for all robots in the system. Therefore, each robot is designed and programmed, prior to deployment, to engage in a specific, unique mission. However, in reality, one may want to deploy a team of robots to possibly multiple missions, while allowing dynamic changes in the robots allocation to different mission as execution progresses.

Consider a disaster-response multi-robot team deployed in a disaster environment. Naturally, we would want the multi-robot team to be *scalable* (allow the deployment a large number robots) and *robust* (easily recover from malfunctions). An additional desideratum that should complement these properties is *dynamism*. For example, the human operator may want all rescue robots to engage in a search and rescue task [Liu and Nejat, 2013], seeking injured or trapped people. During execution a need for a patrol task [Agmon *et al.*, 2011] against further threats may be required. A human operator should be able to partition the robot team into two or more sub-teams, while the robots are operating, and change robots assignments on-the-go as needs change. Note that once a robot is assigned to a team it should automatically adjust its behavior according to the team's mission and coordinate with its new teammates.

In this paper we propose a novel generic architecture and implemented system for allowing multiple robots to dynamically engage in multiple missions with on-the-fly changes. Our framework is *generic*, not domain or platform dependent. It integrates multiple components suggested in the literature alongside newly introduced components resulting an holistic, dynamic, open-source system.

The architecture is based on the fact that one can identify basic abilities of a robot, that are used as building blocks for constructing a whole mission for the robot. When deciding on a new mission on-the-fly, the system injects logic into the composition of those basic abilities, deciding on when to activate what ability. Moreover, this composition is done also in the perspective of a team, namely, deciding when to coordinate, synchronize and allocate tasks and roles for the robot as part of a team. By not hard-coding any characteristics of the mission or the team, it allows those to be changed dynamically, and also easily enables collaboration of heterogeneous robots in a team.

We have implemented the architecture, and report our testings of the system in both simulation and physical deployment that demonstrated its scalability, robustness, efficiency and dynamism. Our code is released for future research.

The remainder of the paper is organized as follows. In Section 2 we survey recent related work. In Section 3 we describe the architecture components, along with the interaction between the layers of the architecture. In Section 4 we describe our implementation of the suggested architecture, as well as demonstrate a real-world physical deployment of our architecture and provide a code-release for our system. Finally, in Section 5 we provide a summary and list future directions for this line of work.

2 Related Work

Being one of the most prominent research areas in robotics, one can find various solutions for handling multi-robot systems. Research on multi-robot systems tend to concentrate on the planning aspects: how to best plan a mission such that the efficiency of the team in task performance is maximized. In such cases, solutions are limited to specific tasks: multi-robot formation [Ren and Sorensen, 2008], multi-robot coverage [Rekleitis *et al.*, 2008], patrolling [Agmon *et al.*, 2012], search and rescue [Liu and Nejat, 2013], and more. Generic solutions concerning challenges that arise from the existence of multiple robots, that are not task-specific, can be found, among others, in handling collision avoidance [Hennes *et al.*, 2012], pathfinding [Standley, 2010], and task allocation [Korsah *et al.*, 2013]. Focus was given also to teamwork infrastructure for software agents, with no implementation on real robotic systems, e.g., [Tambe, 1997].

Bowling et. al. [Bowling *et al.*, 2004] suggest a method for choosing a *play* for a team of soccer-playing robots. The plays are known in advance (given in a form of a *playbook*), and the decision regarding which play to pick is made during the game (similar to choosing a mission during execution). While they offer flexibility in their solution, their work differs considerably from ours, since in their case the team is fixed, and the plays are known prior to the execution.

Software architecture for robot teams offers robustness and flexibility in mission execution by the team. Parker [Parker, 2001] formalized teamwork architecture in ALLIANCE, a framework for controlling a team of robots in a distributed, flexible, way. The architecture was deployed on real robotic systems, showing its advantage in handling events such as robot addition, removal, fault tolerance and more. ALLIANCE was shown to work efficiently in controlling a team of robots performing *one* task (for example, patrolling), limiting its effectiveness to the three lower levels of our suggested architecture: individual, team, and one mission.

Kaminka and Frenkel reported the development of BITE in [Kaminka and Frenkel, 2005], a teamwork architecture that was also implemented in real robotic systems, which offers a flexible, domain-independent teamwork architecture, which aims at minimizing the development effort of such systems, maximizing the reusability of code, and creating a separation between self behavior and team behavior. The programmer inserts *synchronization* points, as well as indication of synchronization and allocation mechanism (there is no commitment in advance to using a specific mechanism).

Similar to ALLIANCE, BITE concentrates on execution of teamwork for a specific task, with a fixed team. MRMM, standing for *Multiple Robots for Multiple Missions*, builds upon teamwork architecture such as BITE, yet allows flexibility in team members and missions by giving the ability to change a mission on-the-fly, move robots from one mission to another, without the need to be in physical proximity to the robots. In addition, it inherently supports a human operator in the loop.

3 Architecture

MRMM proposes a hierarchical architecture consisting of six generic layers. We will first provide an overview of the architecture followed by a more in-depth discussion of each layer.

At the bottom of the hierarchy we have **robots**. The robots can be grouped into teams which in turn can be members of other teams as well. Namely, the robots are abstractly divided into teams and sub-teams, similarly to how a hierarchical organization might operate, in what we call the **team hierarchy** layer. A robot will coordinate its actions and share its knowledge, as needed, with its teammates—the robots that are assigned to the same team as itself. Each (sub-)team can be assigned to a mission which is controlled by a **mission-manager**. The mission manager divides the mission into concrete tasks and generates plans that are assigned to the different teams and their members. It is the mission manager's task to create, oversee and update these plans. Missions are created, updated and terminated by the **system-manager**. The system-manager itself presents all available information regarding the missions and resources, as well as receives its instructions, through a **UI**. However, the communication between human operator(s) (who uses the UI) and the system-manager is mitigated using an intelligent **agent** layer. The agent decides which information to reveal to the human operator(s) and how (e.g., directing the human operator's attention and prioritizing her tasks) and can also take some actions on its own. See Figure 1 for an illustration.

Note that our architecture is not restricted to any given hardware or algorithm used in the different layers.

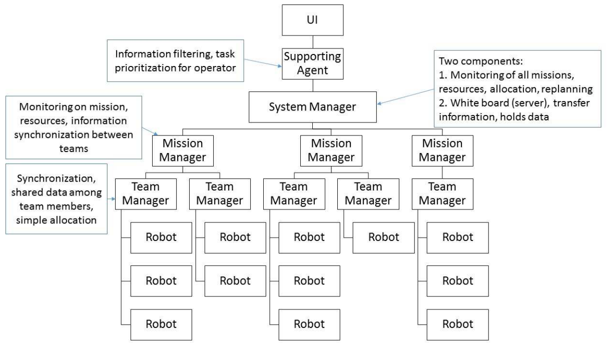

UI
Supporting Agent
Information filtering, task prioritization for operator
Two components:
1. Monitoring of all missions, resources, allocation, replanning
2. White board (server), transfer information, holds data
System Manager
Monitoring on mission, resources, information synchronization between teams
Mission Manager
Mission Manager
Mission Manager
Synchronization, shared data among team members, simple allocation
Team Manager
Team Manager
Team Manager
Team Manager
Team Manager
Robot

Figure 1: The six-layered MRMM architecture components.

3.1 Robots

The bottom layer consists of robots, either homogeneous or heterogeneous, where each of them is mounted with pre-compiled basic functions defined prior to deployment, denoted as *behaviors* - the building blocks of a robotic task. These basic functions can vary between low-level functions (e.g., "turn wheels 20 to the left") and high-level function (e.g., navigate to GPS coordinates x, y, z). The choice of function level depends on the system designer's needs; defining low level functions allows more freedom in generating diverse plans for the robots, yet using high-level functions ease the planning burden. As part of these basic functions a robot can also share information and synchronize its actions with its teammates as necessary.

Each robot operates according to a given *plan*, generated by the mission manager and delivered through the team layer to all team members. A plan is a hierarchical structure which is encapsulated during execution into a sequence of basic function that the robot should execute as discussed below.

3.2 Team hierarchy

In the team hierarchy layer the robots are assigned to one or more teams which in turn can be members of other teams as well (and so on), according to the mission managers instructions. The assignment of robots to teams allows a robot to communicate with only the robots that are relevant to its work. For example, a robot may be planned to synchronize its actions with its teammates. The team hierarchy will allow the robot to execute this synchronization effectively as it is only exposed to messages that are relevant to it. Namely, a plan (which is generated by the mission manager) will use specific teams identities. The team hierarchy can be of any size and shape as needed and instructed by the mission manager.

3.3 Mission Manager

A mission manager is created and overseen by the system manager. The mission manager stores the identity of the resources (e.g., robots, computational resources) that are at its dispense and the mission status at all times. The mission manager is in charge of building the team-hierarchy for its assigned robots and assigns each robot a specific plan. The plans are generated using the notion of *recipes* (similar to those used in STEAM [Tambe, 1997] and BITE [Kaminka and Frenkel, 2005]).

A recipe is a generic plan which needs to be instantiated in order to be executed by a robot. For example, a recipe for a patrol task is originated by the patrol mission planner. It decides, for example, that in order to maximize the frequency of visits along a set of given waypoints, the robots should follow a specific path in a certain pattern. It then creates a recipe for the patrol, which includes the general patrol framework. Since the mission may be executed by heterogeneous robots, it creates a plan per robot type. Namely, the mission manager instantiates the recipe with its knowledge (this is the plan), and assigns the created plans to its robots.

The mission manager is also required to supervise its robots and update the plans during the mission's execution as needed. For example, when a robot loses communication, the mission manager may instantiate new plans for the robot's teammates to avoid obstruction of the rest of the team's work. Similarly, when resources are shifted from one mission to the other by the system manager, the mission manager should be able to adapt the robots' plans to accommodate the changes. The mission manager can also ask the system manager for additional resources when the mission execution falls bellows some defined standards.

The mission manager reports the mission and robots' status to the system manager. Furthermore, the mission manager can be queried by the system manager in order to help the system manager gain better situation awareness. For example, the system manager can ask a mission manager what will be the marginal benefit (in terms of mission fulfillment or performance) from the assignment of an additional resource to the mission.

The mission manager can be mounted on one of the robots (i.e., a robot leader), or on any other device (e.g., a router or a central computer). To prevent the mission manager to become a vulnerability point copies of the mission manager's information is maintained by a sequence of robots (or devices) and a recovery procedure should be defined.

3.4 System Manager

The system manager creates, updates and terminates missions. Specifically, the system manager is aware of all resources in the system and assigns them to the different missions. The manager can change a resource assignment from one mission to another, invoking the mission managers to change the robots' plans.

Once the system manager creates a new mission it assigns a recipe for the mission manager to use. The recipes can be either pre-defined (given by the designer as a "recipe bank") or generated on-line by the system manager itself. In the latter case, the system manager can design a new recipe given a set of requirements by the human operator(s).

The system manager receives information from all mission-managers and passes this information to the agent layer. In the other direction, the system manager receives its instructions (e.g., create a new mission, move a robot from mission x to mission y, etc.) from the agent layer.

3.5 Agent

Supervising and operating multiple robots simultaneously is a difficult task for human operators to do [Goodrich *et al.*, 2005; Chen and Terrence, 2009; Squire and Parasuraman, 2010]. Therefore, the communication process between the human operator(s) and the system manager is mitigated by an automated intelligent agent as recently proposed in [Rosenfeld *et al.*, 2015]. The agent acts as a smart buffer between the system manager and the human operator(s) and performs two main functionalities: First, it filters and highlights some of the information that is passed from the system manager to the UI and thereby directs the human attention towards relevant and critical information. Second, it can take active actions and direct the system-operator to execute different commands (e.g., terminate a mission). For example, when a human operator is engaged in a complex task (e.g., determining a patrol route) the agent should avoid interrupting the operator with uncritical information or requests. In some cases, the agent can provide the needed solution by itself or modify the operator's commands, depending its autonomy description.

The agent should be adaptive to changes in both the environment and the human operator behavior.

3.6 User Interface (UI)

The UI presents the information received from the agent layer to the human operator(s) and passes the instructions given by the human operator(s) to the agent layer. Designing efficient and well manageable UIs for human-multi-robot is necessary in order to ease the human operator's burden and increase the system's performance. This task has attracted a lot of attention over the past years and several proposals have provided intelligent interfaces and interaction modes that have been shown to increase involvement and total human-multi robot team performance [Micire *et al.*, 2011; Clair and Mataric, 2015; Stoica *et al.*, 2013; Sycara and Lewis, 2012].

4 Implementation and Evaluation

We have implemented the MRMM architecture presented in Section 3. In this section we describe the way this was implemented (the code can be found in `www.cs.biu.ac.il/~agmon/MRMM`), and the evaluation of the system in terms of mission efficiency, and coordination overhead.

4.1 Architecture Implementation

The MRMM architecture has been implemented across all six levels. In this section we describe the implemented components, as well as the execution flow of the system, demonstrating the interaction between the levels.

Robots

We used the Hamster robots[1] which use the Robot Operating System (ROS)[2]. *Hamster* is an autonomous unmanned ground vehicle (AUGV) at shelf price of 1600$[3] per robot (See Figure 2). It is a 4WD rugged platform with a built-in navigation algorithm that allows it to explore, map and localize in unknown areas. Hamster has 2 on-board Raspberry PI 3 Linux servers for algorithm execution and an Arduino for low level control. Hamster is mounted with an HD camera with h264 video streaming over WiFi and a 360° 6-meter range LIDAR laser. Each Hamster is 190mm in width, 240mm in length and 150mm in height.

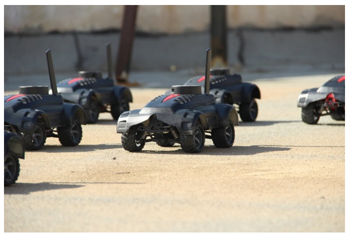


Figure 2: Hamster AUGV; one of the 5 identical robots used in this study.

We defined the hamster's basic functions (behaviors) as follows: "Navigate to point x, y", "Stream video", "Synchronize with teammates".

Team Hierarchy

The team hierarchy is created and updated online by the mission manager and the shared data is updated by the robot members directly.

[1]`http://wiki.ros.org/Robots/Hamster`

[2]`http://www.ros.org/`

[3]2014 pricing.

For simplicity, in the released implementation we assume each mission is conducted by a single team.

Each team has a unique team ID, defined by the system designer. A robot is a team member once it has received instructions from the system manager to join a specific team, and is approved by the mission manager. The team hierarchy is represented in the system in an XML file which maps each team identifier to its robot members and holds the data that is shared among the team members.

When reaching a predefined synchronization point (for example, if robots should enter a room together), a robot announces to the team manager that it is ready for synchronization. Upon receiving all the messages, i.e., all robots in the sub-team are ready, the team manager (through the system manager) announces to all robots that they may proceed. A robot may be requested to synchronize its action with a member of a different sub-team, yet they must share a mission.

Mission Manager

The mission manager manages the different teams and robots (all part of the same mission) using dictionary-based data structures, where all information regarding the mission's robots and status is available. The mission manager queries its robots status every fixed time interval and invokes the planner services in order to instantiate the pre-defined recipes into plans for its robots.

Recipes are represented in our system in XML format and are instantiated to robot plans using designated services that register at the mission manager.

In our implementation, a mission manager is mounted on a router, with the assumption that if the router is down, then the mission is terminated (since in this case the robots cannot communicate, thus will not be able to collaborate as a team). However, the physical location of the mission manager is flexible, and can be placed wherever communication is available between all the mission's team representatives.

System Manager

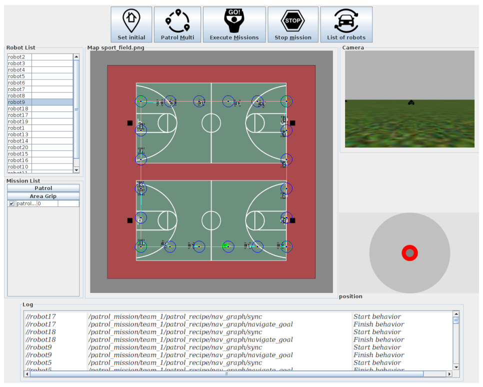
Similar to the mission manager design, the system manager is using dictionary-based data structures where all available resources and active missions are available. Our system includes only two recipes at the moment: Patrolling (e.g., [Agmon *et al.*, 2011]) and static coverage, a.k.a *blanket coverage* (e.g., [Cheng and Savkin, 2013]). In order to add a new recipe, a new planner service needs to be implemented.

The system manager is also mounted on a router, with the assumption that if the central communication mean is lost then the whole system is terminated.

Currently, the system manager simply assigns an equal number of robots to each mission. Implementing a more sophisticated mechanism, either at the Agent layer or the UI layer, is straightforward.

Agent

For simplicity, we implemented a simple agent that does not alter the information provided from system manager to the UI and vice-versa. Instead, we adopted an *advising agent* approach (see [Rosenfeld, 2015]) in which the agent merely observes the environment and the human operator's actions and



```
//robot17   /patrol_mission/team_1/patrol_recipe/nav_graph/sync            Start behavior
//robot17   /patrol_mission/team_1/patrol_recipe/nav_graph/navigate_goal   Finish behavior
//robot18   /patrol_mission/team_1/patrol_recipe/nav_graph/sync            Start behavior
//robot18   /patrol_mission/team_1/patrol_recipe/nav_graph/navigate_goal   Finish behavior
//robot9    /patrol_mission/team_1/patrol_recipe/nav_graph/sync            Start behavior
//robot9    /patrol_mission/team_1/patrol_recipe/nav_graph/navigate_goal   Finish behavior
//robot5    /patrol_mission/team_1/patrol_recipe/nav_graph/sync            Start behavior
```

Figure 3: Screen-shot of the system's GUI, presenting a simulated environment of 20 robots, patrolling along a circular environment.

provides advice as to which task should the operator execute next.

The agent presents the advice in both textual format on the GUI (see Figure 3) as well as in a prerecorded, human-voice message, played in the operator's head-set.

UI

We used a simple GUI as presented in Figure 3. The human operator can select any number of robots using the computer mouse and instruct them to engage in a new mission. In the way, the operator can also shift robots from one mission to another. The UI presents the positioning of each robot and its status.

4.2 Execution Flow

In order to demonstrate the conduct of the system, the responsibilities of each component, and the interaction between them, we now describe the flow of the system in key scenarios: initiating a new mission, removing a robot from a mission, joining a new robot to an existing mission, and switching a robot between two missions.

Initiating a new mission

When a robot is turned on, the only information it must have in advance is the address of the central server (logically, part of the system manager). Once connected to the system manager, it registers to the system by sending its ID, characteristics and capabilities.

When an operator defines a new mission (for example in the patrol mission it defines the waypoints for patrolling), a new mission manager is created, which provides the system manager and the operator information about the mission. This information, along with the details of the robots available for the mission, is sent to the relevant mission planner (here: patrol planner), which is an independent service that can run on any robot or server. The mission planner plans the specific details of the mission (in the patrolling example, it defines the patrolling path and pattern), and sends this plan to the system manager. The system manager sends a plan for each of the robots, while also assigning it to the relevant mission manager. The robot registers to the mission manager, joins the relevant team, and starts executing the mission. The mission manager reports to the system manager upon a successful deployment of the robot to the system.

Removing a robot from a mission

Departure of a robot from a mission can happen in several cases:

- The robot decides it should terminate the mission (commonly happens when the mission is accomplished).
- The robot receives an order to stop performing the mission (for example, when transferring it to a new mission).
- Communication loss between the robot and the mission manager.

Once a mission manager detects that a robot has been (or should be) removed from the mission, it is responsible to notify all relevant teams of its departure, and reports to the operator (via the system manager) of the change in resources to the mission. The system manager decides whether to reassign a new robot to the mission instead of the one that has left, re-plan the mission (by calling the mission planner), or do something else that was defined by the operator (or predefined in the system). Once a new plan is created, it sends it to all members of the relevant mission.

Adding a robot to a mission

A robot may be added to a mission only upon receiving the communication details of the mission manager. Once it is registered to the mission manager, the latter notifies the system manager that it was successfully added to the mission. If the robot does not appear in the current plan, a request for replanning (along with the updated information about the members assigned to the mission) is sent to the mission planner. Once receiving a new plan, it is sent to all robots in the mission (including the new robot assigned to the mission).

Switching a robot between missions

When the operator or the system manager decides to move a robot from one mission to another, the robot receives a command to stop the mission, along with a single piece of information: the details of the new mission manager the robot has to connect to. Upon receiving this command, the robot leaves the mission, registers to the new mission manager, and waits for a plan. The process of joining a mission has been described above.

4.3 Deployment and Evaluation

The system has been fully implemented on four Hamster robots in a patrolling mission. In addition, excessive experiments have been conducted in simulation, using ROS/Gazebo simulated environment. The goal of the experiments was threefold: Perform a sanity check for the execution flow and the behavior of the system, examine the coordination cost of the system, and evaluate the performance of the missions. Concentrating one main mission—Multi Robot Patrolling—and one minor one—Blanket Coverage—the system was shown to pass successfully the sanity check, have little overhead in terms of communication cost associated with the system management, and increase efficiency of the system by using teams of robots. In all experiments, there was no need to restart the robots when switching to a new mission, and it was possible to instantiate the missions on-the-fly, as designed originally.

The patrolling algorithm used was the circular patrol. In this scenario, the n patrolling robots are spread uniformly (in time) along the patrol path, and maintain this distance between them throughout the execution.

Experiments in a physical system

The system consisted of four Hamster robots, all performing a patrolling task in an outdoor environment (a basketball court)[4]. In the experiment, we demonstrated the behavior of the system in the following flow of execution:

1. Robots are turned on, connect to the system manager, and wait for instructions.
2. The operator decides to execute a patrolling task, and defines the waypoints for the mission.
3. The patrol-planner is launched, and creates a patrol path through the waypoints using a TSP approximation [Agmon *et al.*, 2011].
4. The plan is sent to the robots, and the allocation of initial location for each robot is decided based on the Hungarian Algorithm.
5. Communication with the centralized station is turned off. The robots continue to perform the patrolling task, as expected. This shows that the system can be distributed, thus needs no centralized control.
6. A robot is taken out of the system. The team realizes that the robot has exited, the system manager displays this information, and the remaining team continues to perform without the extracted robot (specifically, they do not get stuck and wait for it indefinitely).
7. The extracted robot returns to the system, and joins the team/mission again.

The experiment completed successfully, which shows the flexibility and dynamic nature of the system.

Simulated environment

We have run extensive experiment in ROS/Gazebo simulated environment with $5, 10, 15$ and 20 robots. Two scenarios were tested:

- Robots performing the patrolling mission.
- Robots performing the patrolling mission, and after one round of patrol, 5 of them were extracted to the blanket-coverage mission.

Each patrolling mission was executed for 10 rounds (each experiment lasting between 40 to 72 minutes).

Figure 4 describes the number of messages exchanged in the system throughout the execution of the patrolling problem. At each synchronization point, the leading robot is responsible for the synchronization: it receives a synchronization message from all teammates, and distributes a *synched* message to all. Thus the number of messages exchanged in each point is linear in the number of robots. However, the more robots there are in the system, we must add more synchronization points (note that synchronization points are not necessarily the waypoint). thus the total number of messages grows, and is polynomial in the number of robots. We also show in the figure the number of messages exchanged in the system in case 5 robots leave the team to perform a new task (the notion $x \rightarrow y$ denoted a case in which x robots started the experiment, and a after one rounds, 5 robots are extracted to the new mission). They all engage in a cooperative patrolling task, and after one round - the group of 5 leave the team, leaving the rest of the robots to patrol. In each pair of values - on the right: number of messages sent in an eventless execution of $5, 10,$ and 15 robots. On the left - the number of sent messages if we start with $10, 15$ and 20 robots, and 5 of them are taken to a new mission soon after the experiment started. Surprisingly, the number of synchronization messages when removing 5 robots to a new mission is slightly (insignificantly) smaller than the number of synchronization messages when they are assigned to the mission from the beginning. The reason lies in the points they were extracted from: when replanning after the removal of 5 robots, the new chosen synchronization points were closer to the waypoints, thus required less synchronization throughout the execution. We leave it for future work to optimize synchronization points also with respect to the synchronization cost, and not only with respect to the performance of the mission itself (here, idleness criteria in the patrolling mission).

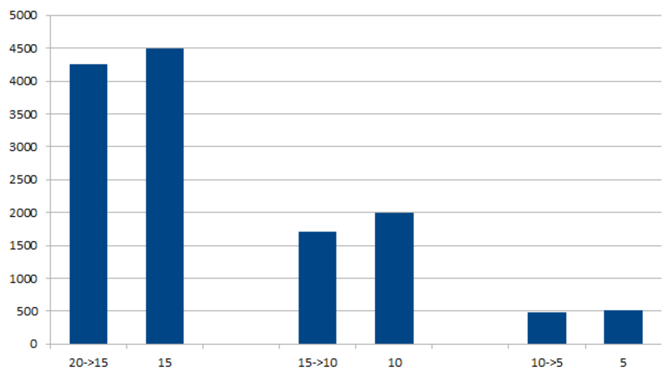


Figure 4: The number of messages sent in the system by a team of 10, 15 and 20 robots, where 5 are extracted to a new mission after one round of patrolling.

Figure 5 describes the *idleness* criteria, which corresponds to the performance of the mission by the team of robots. It is clear that as we add more robots to the system, the performance of the system grows. Here: the idleness (time between visits to a point) decreases. It is interesting to see that after reaching 15 robots, there is a plateau in the performance, i.e., adding more robots does not increase the productivity of the system (the idleness does decrease). This happens because of the overhead spent on synchronization (see this data above each bar in the figure). As the number of robots grow, cost of coordination (measured in coordination time) increases.

[4]Summary of the experiment can be seen here: `https://www.youtube.com/watch?v=oLF89tekvmI`

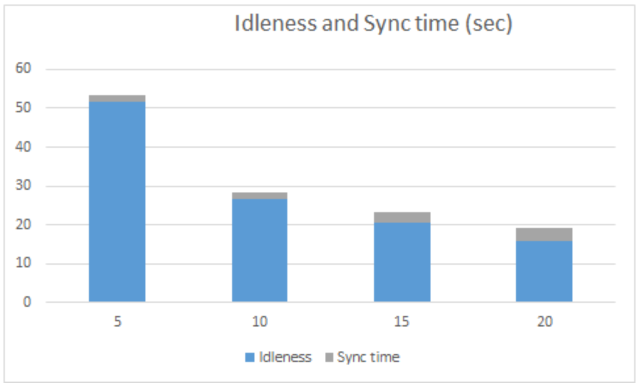


Figure 5: The idleness criteria as the number of robots grow. Above: the average time spent on synchronization.

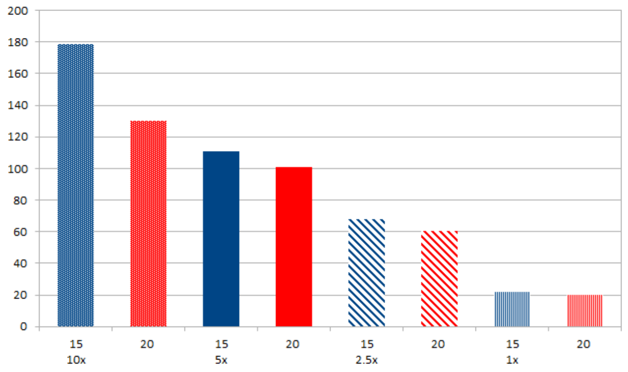


Figure 6: The resulting idleness obtained by a team of 15 and 20 robots. Increasing the size of the space in factors of 2.5, 5 and 10. In each pair or values - on the right: 20 robots, and on the left: 15 robots.

However, if the environment grows, thus the time between visits (idleness) grows as well, the relative cost of synchronization becomes negligible compared to the idleness (see Figure 6), thus losing the plateau effect (or more accurately: shifting it to larger values of number of robots).

5 Conclusions

In this paper, we have presented a novel architecture for controlling a large group of heterogeneous robots, responsible for carrying out multiple missions by multiple sub-teams of robots. The layers span from the robotic platform (in the "lower level" of the system) to a human operator at the top of the system, possibly being assisted by a supporting agent. The missions are saved as recipes, instantiated for the specific type of (possibly heterogeneous) robots available in the system by a *plan*. The operator can create new recipes for new types of robots, and send them the plans on-the-fly. We have implemented the system, and have shown its effectiveness in handling teams of robots engaged in different missions, in the same physical compound.

This architecture, and its implementation (available online), can be used in various multi-robot systems, thus creates many new opportunities for future research - both on the system itself, and by using the system. Among those one can consider increasing efficiency of the system in terms of communication constraints, allowing a robot to be part of two completely different missions, and more.

References

[Agmon *et al.*, 2011] Noa Agmon, Daniel Urieli, and Peter Stone. Multiagent patrol generalized to complex environmental conditions. In *AAAI*, 2011.

[Agmon *et al.*, 2012] Noa Agmon, Chien-Liang Fok, Yehuda Emaliah, Peter Stone, Christine Julien, and Sriram Vishwanath. On coordination in practical multi-robot patrol. In *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 650–656, 2012.

[Ardiny *et al.*, 2015] Hadi Ardiny, Stefan John Witwicki, and Francesco Mondada. Are autonomous mobile robots able to take over construction? a review. *International Journal of Robotics*, 4(EPFL-ARTICLE-217004):10–21, 2015.

[Auat Cheein and Carelli, 2013] Fernando Alfredo Auat Cheein and Ricardo Carelli. Agricultural robotics: Unmanned robotic service units in agricultural tasks. *Industrial Electronics Magazine, IEEE*, 7(3):48–58, 2013.

[Bowling *et al.*, 2004] Michael H Bowling, Brett Browning, and Manuela M Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 376–383, 2004.

[Chen and Terrence, 2009] JYC Chen and PI Terrence. Effects of imperfect automation and individual differences on concurrent performance of military and robotics tasks in a simulated multitasking environment. *Ergonomics*, 52(8):907–920, 2009.

[Cheng and Savkin, 2013] Teddy M Cheng and Andrey V Savkin. Decentralized control of mobile sensor networks for asymptotically optimal blanket coverage between two boundaries. *IEEE Transactions on Industrial Informatics*, 9(1):365–376, 2013.

[Clair and Mataric, 2015] Aaron St Clair and Maja Mataric. How robot verbal feedback can improve team performance in human-robot task collaborations. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 213–220. ACM, 2015.

[Dasgupta *et al.*, 2015] Prithviraj Dasgupta, José Baca, KR Guruprasad, Angélica Muñoz-Meléndez, and Janyl Jumadinova. The comrade system for multirobot autonomous landmine detection in postconflict regions. *Journal of Robotics*, 2015, 2015.

[Goodrich *et al.*, 2005] Michael A Goodrich, Morgan Quigley, and Keryl Cosenzo. Task switching and multi-robot teams. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 185–195. Springer, 2005.

[Guizzo, 2008] Erico Guizzo. Three engineers, hundreds of robots, one warehouse. *Spectrum*, 45(7):26–34, 2008.

[Hennes *et al.*, 2012] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 147–154, 2012.

[Kaiman, 2016] Jonathan Kaiman. At japan's fukushima nuclear complex, robots aiding the cleanup after 2011 disaster, March 2016. [Online; posted 10-March-2016].

[Kaminka and Frenkel, 2005] Gal A Kaminka and Inna Frenkel. Flexible teamwork in behavior-based robots. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 20, page 108, 2005.

[Korsah *et al.*, 2013] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[Kulkarni and Pompili, 2010] Indraneel S Kulkarni and Dario Pompili. Task allocation for networked autonomous underwater vehicles in critical missions. *Selected Areas in Communications*, 28(5):716–727, 2010.

[Liu and Nejat, 2013] Yugang Liu and Goldie Nejat. Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent & Robotic Systems*, 72(2):147–165, 2013.

[Micire *et al.*, 2011] Mark Micire, Eric McCann, Munjal Desai, Katherine M Tsui, Adam Norton, and Holly A Yanco. Hand and finger registration for multi-touch joysticks on software-based operator control units. In *Technologies for Practical Robot Applications (TePRA)*, pages 88–93. IEEE, 2011.

[Parker, 2001] Lynne E Parker. Evaluating success in autonomous multi-robot teams: experiences from alliance architecture implementations. *Journal of Experimental & Theoretical Artificial Intelligence*, 13(2):95–98, 2001.

[Rekleitis *et al.*, 2008] Ioannis Rekleitis, Ai Peng New, Edward Samuel Rankin, and Howie Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):109–142, 2008.

[Ren and Sorensen, 2008] Wei Ren and Nathan Sorensen. Distributed coordination architecture for multi-robot formation control. *Robotics and Autonomous Systems*, 56(4):324–333, 2008.

[Rosenfeld *et al.*, 2015] Ariel Rosenfeld, Noa Agmon, Azaria Amos Maksimov, Oleg, and Sarit Kraus. Intelligent agent supporting human-multi-robot team collaboration. In *Proceedings of the Twenty-Fouth International Conference on Artificial Intelligence (IJCAI)*, 2015.

[Rosenfeld, 2015] Ariel Rosenfeld. Automated agents for advice provision. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 4391–4392. AAAI Press, 2015.

[Saez-Pons *et al.*, 2010] Joan Saez-Pons, Lyuba Alboul, Jacques Penders, and Leo Nomdedeu. Multi-robot team formation control in the guardians project. *Industrial Robot: An International Journal*, 37(4):372–383, 2010.

[Squire and Parasuraman, 2010] PN Squire and R Parasuraman. Effects of automation and task load on task switching during human supervision of multiple semi-autonomous robots in a dynamic environment. *Ergonomics*, 53(8):951–961, 2010.

[Standley, 2010] Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 1, pages 28–29, 2010.

[Stoica *et al.*, 2013] Atanasia Stoica, Theodoros Theodoridis, Huosheng Hu, Klaus McDonald-Maier, and David F Barrero. Towards human-friendly efficient control of multi-robot teams. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 226–231. IEEE, 2013.

[Sycara and Lewis, 2012] Katia Sycara and Michael Lewis. Human control strategies for multi-robot teams. In *Proceedings of the 16th WSEAS International Conference on Computers*, pages 149–154. World Scientific and Engineering Academy and Society, 2012.

[Tambe, 1997] Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence*, pages 22–28, 1997.