Multi-Robot Task Acquisition through Sparse Coordination

Guglielmo Gemignani[1⋆†], Steven D. Klee[2⋆], Daniele Nardi[1], and Manuela Veloso[2]

***Abstract*— In this paper, we consider several autonomous robots with separate tasks that require coordination, but not a coupling at every decision step. We assume that each robot *separately* acquires its task, possibly from different providers. We address the problem of multiple robots *incrementally* acquiring tasks that require their *sparse-coordination*.**

To this end, we present an approach to provide tasks to multiple robots, represented as sequences, conditionals, and loops of sensing and actuation primitives. Our approach leverages principles from sparse-coordination to acquire and represent these joint-robot plans compactly. Specifically, each primitive has associated preconditions and effects, and robots can condition on the state of one another. Robots share their state externally using a common domain language. The complete sparse-coordination framework runs on several robots. We report on experiments carried out with a Baxter manipulator and a CoBot mobile service robot.

I. INTRODUCTION

Multi-robot systems are often more reliable, affordable and fault tolerant than single robots. However, they require complex coordination mechanisms to be effective. Robots can coordinate using a wide variety of techniques, ranging from completely centralized to distributed approaches. There are also many means to provide tasks to robots, including programming each robot, providing goals in a domain to a multi-agent planner, and directly teaching the robots.

In this work, we consider the problem of *separately* and *incrementally* providing tasks to a group of autonomous robots that need to infrequently coordinate. The robots may have very different internal representations of their state, actuation capabilities, and sensing capabilities. Furthermore, they may acquire their tasks in different manners. For example, a manipulator may be taught by a human through natural language, and a mobile base may acquire tasks from a planner. However, to pick up a package and deliver it, the two robots must work together. In these kinds of tasks, we note that the robots do not need to coordinate at every decision step. In fact, much of their tasks can be completed entirely independently. In literature, this concept of coordinating when necessary is known as *sparse-coordination* [1]. In terms of task representation, sparse-coordination represents the joint state space only when the robots need to coordinate.

[1]Guglielmo Gemignani and Daniele Nardi are with the Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Via Ariosto, 25 00185 Rome, Italy. {gemignani,nardi}@dis.uniroma1.it

[2]Steven D. Klee and Manuela Veloso are with the Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA sdklee@andrew.cmu.edu, mmv@cs.cmu.edu

⋆ The first two authors contributed equally to this work.

† Guglielmo Gemignani contributed to this work while visiting Carnegie Mellon University.

Fig. 1: Baxter manipulator and CoBot mobile service robots coordinated with the proposed approach.

Our goal is to enable heterogeneous robots, acquiring tasks from different providers, to solve problems requiring sparse coordination. We first note that coordinating different robots requires a common means of communication. To this end, we contribute a task representation for a robot to incrementally acquire a task with preconditions and effects represented in a shared domain language. Then, we present an approach to sparsely coordinate robots using this task representation. Specifically, each agent conditions on the state of other robots to sparsely coordinate.

Tasks are represented in graph-based structures composed of action and sensing primitives, conditionals, and loop structures. The preconditions and effects of the actions are written using a common domain language. Robots keep track of their own state, and condition on the state of each other by sending queries to interact. By only representing the coordination between robots when necessary, this approach is partially immune to the combinatorial explosion in the number of states found in other representations.

Our contribution has been used to coordinate several robots, including a Baxter and a CoBot robot. Baxter is an industrial manipulator robot able to perform complex manipulation tasks. CoBot is instead an omnidirectional mobile service robot equipped with a variety of sensors including a laser range finder, microphones, a camera, and Microsoft Kinect sensors [3]. We first show an example of how the two arms of a Baxter manipulator can be treated as two autonomous agents and coordinated. We then show a more complicated example involving a CoBot and the two arms. Figure 1 shows both robots.

In the next section, we present an overview of related work, with a focus on past research in representing multi-agent plans compactly and on providing tasks incrementally to a robot. We then introduce the technical details of our approach, present demonstrative examples, and conclude with a summary of the contributions and a brief discussion of future work.

II. RELATED WORK

Our work is mainly related to the literature on multi-agent planning and incremental task acquisition. Multiple approaches have been proposed to represent multi-agent plans. For example, there are several techniques to subdivide joint tasks into smaller tasks that each agent can execute autonomously or as part of a smaller group. These techniques can rely on communication models [4], on dedicated architectures [5], [6], and on collections of conventions followed by all team members [7]. Additionally, Coordination Graphs compactly represent dependencies between the actions of different agents, thus capturing the local interaction between them [8]. Local interactions have also been exploited to minimize communication overhead during policy execution [9] and in game-theory to obtain compact game representations [10]. These approaches rely on dependency analysis to decompose independent parts of the initial representation. Finally, Petri Net Plans are another framework for collaboration, similar in principle to our approach [11]. However, the plans are not acquired through user instruction, and there is no query language for checking the state of other robots. All of these approaches require that a complete plan is given to each robot a priori. In our work, tasks are acquired by the robot incrementally, and the provider specifies the dependencies between different robot tasks.

Research also addresses the problem of acquiring a task incrementally, focusing on composing robot primitives into different task representations. An early approach that tackles such a problem focuses on creating sequences of robot primitives, which represent its action and sensing capabilities [12]. To support conditionals, tasks have also been represented as acyclic graphs composed of nodes representing finite state machines [13]. Tasks have also been represented as formal logic goal descriptions [14], where the instructions received from the provider are associated with formal logic expressions that represent the task provided to the robot.

Recently, a more expressive framework based on Instruction Graphs has been proposed to support teaching tasks with loops and conditionals [15]. Other works have proposed representations of parametric tasks that are defined by the provider at an abstract level [16]. Neither of these representations keep track of the state of the robot. In more recent work, robot action and sensing primitives have been associated with preconditions and effects [17], [18]. With this additional information, the robot can propose goal-oriented plans, instead of purely acquiring a task step-by-step.

All of these works on task acquisition focus on a single robot, not addressing the problem of incrementally providing tasks to multiple coordinating robots. In this work, we focus on robots that sparsely-coordinate [1]. Sparsely coordinated robots have tasks that require infrequent cooperation, not at every decision step.

In the next section, we build upon the Instruction Graph framework, first adding robot-primitive preconditions and effects. Then, we allow robots to condition on the state of each other to support sparse interactions.

III. APPROACH

We consider several autonomous robots with primitives that represent their actions and sensing capabilities. Each robot acquires its task separately and incrementally by interacting with a provider, be it a user or a planner. Our goal is to allow these robots to sparsely coordinate to complete their tasks. In this section, we first describe our original task teaching framework for individual robots. Then, we present a detailed description of our task representation that encapsulates the robot state, and our approach for multi-agent sparse coordination.

A. Instruction Graphs

For an individual robot, not interacting with others, we represent tasks as Instruction Graphs (IG) [15]. Instruction Graphs are graphs where vertices represent robot-primitives, while edges represent possible transitions between vertices. Formally, an Instruction Graph is a graph $G = \langle V, E \rangle$. Each vertex $v \in V$ is a tuple:

$$v = \langle id, InstructionType, f, P \rangle$$

where *id* is an integer, *InstructionType* is the type of the vertex and f is a function with a set of parameters P. The function f represents an action or sensing primitive that the agent should perform when visiting the vertex.

Each Instruction Graph is executed starting from an initial vertex, until a termination condition is reached. During execution, the *InstructionType* of the vertex describes how the robot should transition to the next vertex based on the output of the function f. The IG framework defines the following *InstructionTypes*:

- **Do and DoUntil**: Used for sequences of primitives. f has no output, and the algorithm transitions along the sole out-edge. Here, we will refer to both of these types of nodes simply as Actions.
- **Conditionals**: Used for sensing actions. The algorithm interprets f as a boolean value used to transition to one of two children.
- **Loops**: Used for looping structures. The algorithm interprets f as a boolean value, and vertices inside of the loop are repeated while the condition is true.

Figure 2 shows an example node with id 2, *InstructionType* Action, function *move_forward*, and parameter *5 meters*. This corresponds to the second node in an IG, which executes a robot primitive to move a mobile base forward 5 meters. We refer the reader to the original work on Instruction Graphs for a more detailed overview [15].

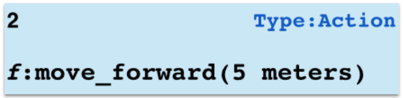


Fig. 2: Example of IG node with id 2, *InstructionType* Action, function *move_forward*, and parameter *5 meters*.

Instruction Graphs are incrementally constructed through the interaction with a user. Specifically, natural language commands are processed by a probabilistic parser and grounder [20]. This allows the robot to learn the groundings from natural language to robot primitives, environmental features, and tasks. The robot starts with an initial knowledge base of groundings and learns more over time by asking the user when it is unsure of how to ground an expression.

B. Sparse-Coordination Instruction Graphs

To sparsely coordinate, robots must keep track of their state and be able to query the state of one another. We define Sparse-Coordination Instruction Graphs (SCIG) as graphs $G = \langle V, E \rangle$ where each vertex v is a tuple:

$$v = \langle id, InstructionType, f, P, Prec, Eff \rangle$$

where the additional elements *Prec* and *Eff* respectively represent sets of preconditions and effects of the function *f*. More generally, each function f has an associated set of literals $\mathscr{L}_f$ that represents its preconditions and effects. Thus, we define:

$$\mathscr{L} = \bigcup_{\forall f} \mathscr{L}_f$$

as the common domain language used by all of the robots. We represent each literal using STRIPS semantics [2]. In particular, each action adds or removes positive literals from the robot's current state. While robots may represent their internal state differently, their primitives express this state in terms of the common set of strips literals, $\mathscr{L}$.

To associate these preconditions and effects to actions, each robot sensing and actuation primitive is defined in the Planning Domain Definition Language (PDDL) [19].[1] For example, a Baxter manipulator may have an action *pick_up(object_id)*, to pick up an object with a given ID. Internally these objects are represented as a 3D point in space and bounding boxes. However, the effects of the action are to remove the literal *hand_empty*, and then add the literal *holding(object_id)*. Figure 3 shows an example PDDL definition for Baxter's "pickup" action. Currently, we assume that all changes in state are captured by the robot primitives and that each robot can only modify its own state.

```
(:action pickup
 :parameters (?x)
 :precondition (and (OBJECT ?x)
                    (hand_empty))
 :effect       (and (holding ?x)
                    (not (hand_empty)))
```

Fig. 3: Example PDDL definition for the primitive "pickup". As preconditions, its parameter must be an object, and the hand must be empty. The effects are that the hand is no longer empty, and the robot is holding an object.

During execution, each robot keeps track of its own state. Specifically, the state predicates are either appended or deleted from the robot's state according to the effects of the executed action. We introduce a special function *check_literal*, used in Conditional and Looping vertices that can condition on the state of any agent. The *check_literal* function takes as input a unique robot identifier and a query. In our framework the query is represented as a set of STRIPS predicates, possibly composed with the *and*, *or* and *not* operators. The query is routed to the the robot with the corresponding identifier.

When a robot receives a query, it is evaluated against its current state. Each robot adopts a closed-world assumption when responding to queries. In particular, the robot checks that positive literals are present in its state and that negated literals are absent in its state. The result of this query is returned to the requesting robot. In this way each robot has only a representation of its own state, and makes no assumptions about the state of another.

Figure 4 shows a partial example of a SCIG for a CoBot mobile base, where the *check_literal* function is used to condition on the state of a Baxter manipulator. Specifically, the CoBot will perform the *move_to* action if Baxter's state does not contain *hand_empty*.

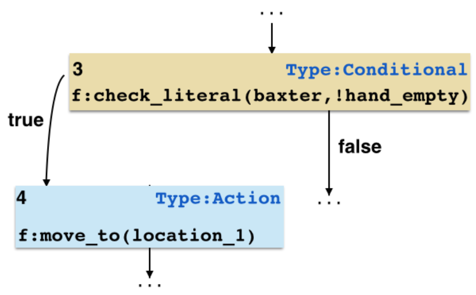


Fig. 4: Partial example of a SCIG for a CoBot conditioning on the state of a Baxter manipulator. If Baxter's state does not contain *hand_empty*, CoBot will perform the *move_to* action.

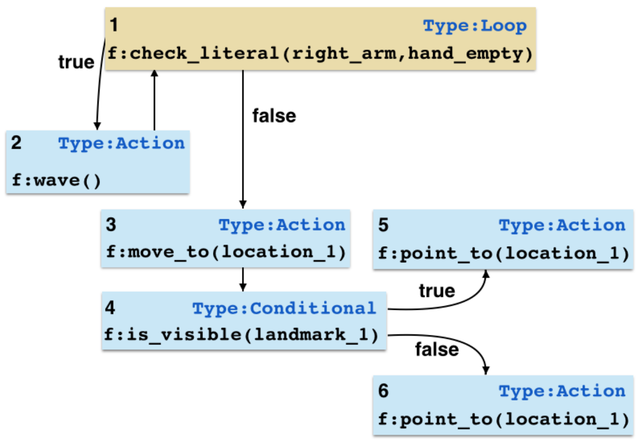
With this approach we are able to implement coordination at a high level. In particular, we define several useful coordination actions from Loops and Conditionals:

- **Wait Until**: The robot waits until another robot is in some state. This is implemented with a Loop.
- **Act Until**: The robot repeats some actions until another robot is in some state. This is implemented with a Loop.
- **Ask**: The robot conditions on the state of another robot. This is implemented with a Conditional.

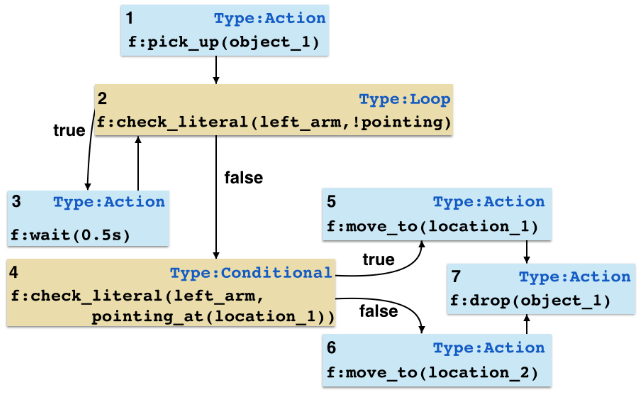
We provide examples of each of these forms of coordination in the next sections.

We note that for sparse-coordination, many of typical problems of multi-robot communication do not arise. For instance, consistency is not an issue, because the robots keep track of only their own state, and directly query each other's state as needed. Since coordination is infrequent, and at a high-level, the robots can also cooperate in environments with high-latency and low-bandwidth. In this work, we do not address the problem of faulty sensors or non-deterministic action effects. For now, we assume that the robot-primitives are all fault tolerant.

[1]Although we represent actions using PDDL, any other language could be used to define the common domain language.



(a)



(b)

Fig. 5: Sparse-Coordination Instruction Graphs extracted from the text in Figure 6. Specifically, (a) represents the graph for the left arm, while (b) represents the graph for the right arm. The nodes in yellow represent the vertices that require a query to another robot's state.

IV. Demonstrative Examples

We coordinated several robots with the presented approach, including a manipulator and a mobile base. In this section we show how multiple robots can be taught to perform different tasks that involve sparse-coordination. In particular, we show how a Baxter robot can be instructed to perform a manipulation task during which its two arms need to sparsely interact with each other. Then, we extend this example by considering also a CoBot mobile base. In this case, we show how the two arms and the mobile service robot can be instructed to deliver and store an object.

A. Store Task

We first show how the two arms of a Baxter manipulator can be treated as separate agents and coordinated. We denote the arms as *left_arm* and *right_arm*. Table I shows the set of robot primitives for both arms, with their associated preconditions and effects. Our task is to have the left arm assist the right arm in finding an unobstructed location to place an object.

TABLE I: Baxter arms primitives with associated preconditions and effects.

Action Primitives	Preconditions	Effects
wave()	-	-
wait(time)	-	-
is_visible(landmark_id)	-	-
move_to(location)	-	-pointing, -at(old_location), +at(location)
pick_up(object_id)	hand_empty	-hand_empty, +holding(object_id)
drop(object_id)	holding(object_id)	-holding(object_id), +hand_empty
point(location_id)	-	-at(old_location_id), +at(location_id), +pointing, +pointing_at(location_id)

```
Left arm:

wave while right hand is empty
move to location 1
if landmark 1 is visible
point to location 1
otherwise point to location 2

Right arm:

pick up object 1
wait while left arm is not pointing
if left arm is pointing at location 1
move to location 1
otherwise move to location 2
end if
drop object 1
```

Fig. 6: Natural language input provided to the two arms. The names of the agents are shown in red, and their states are shown in blue.

In this example, a user describes the task to each agent in two separate teaching sessions through natural language. Specifically, the left arm is instructed to wave until the right arm picks up an orange wooden block (Figure 9a). At this point, the state of the right arm is changed to *holding(object_1)* and the function *check_literal(right_arm, hand_empty)* returns false. After realizing this fact, the left arm starts checking if a landmark can be detected at the drop position (Figure 9b). In the case a landmark is detected, the left arm points at it, reaching the *pointing_at(location_1)* state. The function *check_literal(left_arm, !pointing)* now returns false and the right arm drops the block at *location_1* (Figure 9c). Instead, when the landmark is not detected the left arm points at an alternative location (*location_2*) where the orange block can be dropped (Figure 9d).

Figure 6 shows a natural language description of the task provided to the two arms. Instead, Figures 5a and 5b show the corresponding Sparse Coordination Instruction Graphs.

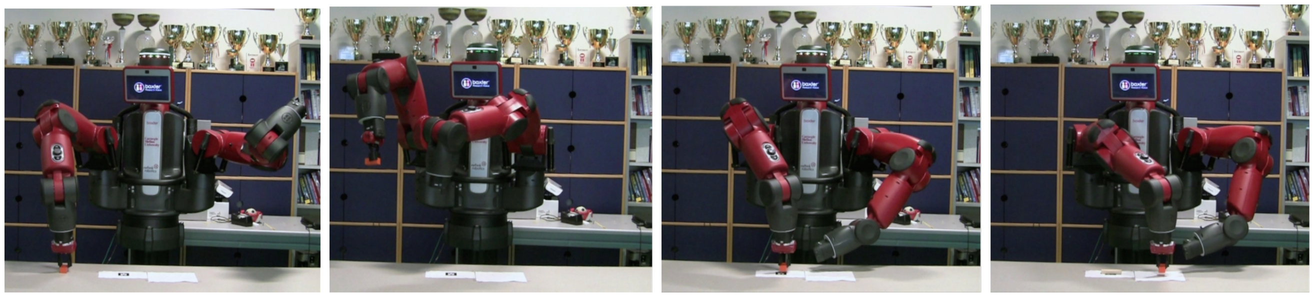

(a) (b) (c) (d)

Fig. 7: (a) The left arm is shown waving until the right arm picks up an orange wooden block. (b) The left arm checks if the drop position is open. (c) Since this position is open, the left arm points at it where the object will be placed. (d) In the second run, since the drop position is not open, the left arm points at an alternative position.

In this specific case, the natural language description of the tasks was parsed through the aid of specifically developed parsers, similarly to a previous work [20]. The descriptions were grounded to objects and locations of a knowledge base containing a high-level description of the environment and the robot primitives. In this example we assume that the two robots have the same high-level representation of the environment. In other words, we assume that the two robots agree on the position of the two possible locations. The execution of the task can be seen in the video attached.[2]

B. Deliver and Store Task

Next, we extend the previous example by considering also a CoBot robot [3]. CoBot must coordinate with the Baxter's arms in order to deliver and store an object. To this end, we modify the previous example by making the *right_arm* wait for CoBot to be at a specific location. Note that the *right_arm* is instructed to wait just for simplicity. It could also be instructed to do other work while waiting for CoBot's arrival. When CoBot reaches the state *at(location_3)*, the *right_arm* will pick up the delivered object from CoBot's basket, storing it at the location pointed out by the *left_arm*. For CoBot, we defined the robot primitives shown in table II. For Baxter we used the previously described primitives.

TABLE II: CoBot primitives with associated preconditions and effects.

Action Primitives	Preconditions	Effects
Say(message)	-	-
Move_to(location)	-	-at(old_location_id), +at(location_id)

The user teaches the tasks to the robots in three separate teaching sessions. Figure 8 shows a natural language description of the tasks provided to CoBot and to the *right_arm*. The task given to the *left_arm* is the same as in the previous example. Figure 10 depicts the Sparse Coordination Instruction Graph extracted for CoBot. Since the SCIG of the right arm is almost identical to Figure 5b, we omit it due to space constraints.

[2]http://youtu.be/4nLjuLhFUvk

```
CoBot:

move to location 3
say ``I am here to deliver a package"
wait while left arm is not pointing
move to location 4

Right arm:

wait while CoBot is not at location 3
pick up object 1
wait while left arm is not pointing
if left arm is pointing at location 1
move to location 1
otherwise move to location 2
end if
drop object 1
```

Fig. 8: Natural language input provided to CoBot and the *right_arm*. The description of the *left_arm* task instead is the same shown in Figure 6. The names of the robots are shown in red while their states are shown in blue.

C. Discussion

Sparse-Coordination Instruction Graphs allow users to teach a wide-variety of tasks that require multi-agent coordination. In particular, they are well suited to tasks that require high-level cooperation between robots. We have found the approach especially effective with robots that have separate goals. For instance, our fleet of CoBots perform many tasks, such as escorting people and picking up objects. Some of these tasks require brief interaction with Baxter, which has its own goals to accomplish. The robots coordinate infrequently because their goals require a limited amount of interaction.

We can also represent joint-plans with goals that require a tight coupling of robot actions at each decision step. An example of such a task is the bimanual manipulation of a large object. However, each robot will need to make many queries to represent most of the joint-state space before acting. Thus, it is often impractical for a user to teach tightly coordinated tasks to the robot. Currently, tightly-coordinated tasks can be taught to the robots using a planner.

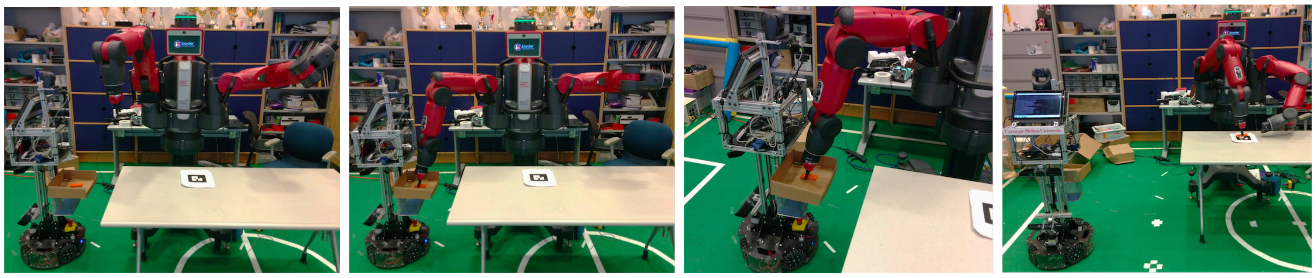


Fig. 9: (a) CoBot reaches the pick up location while the right arm is waiting and the left arm is waiving. (b) The right arm can now pick up the object. (c) Detailed view of Baxter picking up the object. (d) The left arm points at location 1, the right arm drops the object, and CoBot leaves.

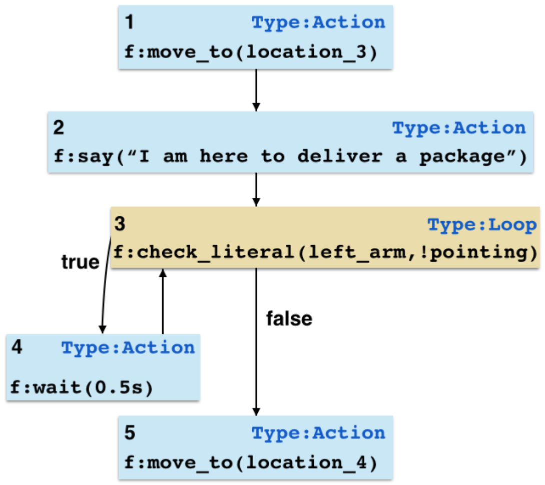


Fig. 10: SCIG extracted from the CoBot's task description in Figure 8. Node 3 requires a query to another robot's state, in this case Baxter's left arm.

V. CONCLUSION

In this paper, we addressed the problem of multiple robots separately and incrementally acquiring tasks that require coordination. By leveraging principles from sparse-coordination we enable robots to acquire and represent joint-robot plans compactly. Specifically, we introduced Sparse-Coordination Instruction Graphs, which encapsulate robot-primitive preconditions and effects. The robots act independently, and only coordinate when necessary by querying each other's state. We demonstrated this approach with two examples. First we treated both arms of a Baxter robot as separate agents and had them coordinate to store an object at an unobstructed location. Then, we extended this example by having a CoBot mobile base deliver the object that the arms stored.

As a future work, we are investigating extensions to represent tasks that require tight coordination more compactly. We are also interested in how our coordination approach can be used with knowledge-acquiring actions. For instance, in cloud robotics a robot-primitive may request information, or even queue a task on another robot.

REFERENCES

[1] F. S. Melo and M. Veloso, "Decentralized MDPs with Sparse Interactions", in Artificial Intelligence, 2011.
[2] R. E. Fikes, and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", in Artificial Intelligence, 1972.
[3] J. Biswas and M. Veloso, "Localization and Navigation of the CoBots Over Long-Term Deployments", in The International Journal of Robotics Research, 2013.
[4] B. P. Gerkey, and M. J. Matarić. "Sold!: Auction Methods for Multirobot Coordination", in IEEE Transactions on Robotics and Automation, 2002.
[5] L. E. Parker, "ALLIANCE: An Architecture for Fault Tolerant Multi-robot Cooperation", in IEEE Transactions on Robotics and Automation, 1998.
[6] M. Tambe, "Agent Architectures for Flexible, Practical Teamwork", in Artificial Intelligence, 1997.
[7] P. Stone, "Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer", MIT Press, 1998.
[8] J. Kok, P. Hoen, B. Bakker, and N. Vlassis, "Utile Coordination: Learning Interdependencies Among Cooperative Agents", in IEEE Symposium on Computational Intelligence and Games, 2005.
[9] M. Roth, R. Simmons, and M. Veloso, "Exploiting Factored Representations for Decentralized Execution in Multiagent Teams", in AAMAS, 2007.
[10] J. A. Xin, K. Leyton-Brown, and N. Bhat, "Action-Graph Games", in Games and Economic Behavior, 2008.
[11] V. A. Ziparo, L. Iocchi, P. Lima, D. Nardi, and P. Palamara, "Petri Net Plans - A Framework for Collaboration and Coordination in Multi-Robot Systems", in AAMAS, 2011.
[12] S. Lauria, G. Bugmann, T. Kyriacou, J. Bos, and E. Klein, "Personal Robot Training via Natural-Language Instructions", in IEEE Intelligent Systems, 2001.
[13] P. E. Rybski, K. Yoon, J. Stolarz, and M. Veloso, "Interactive Robot Task Training Through Dialog and Demonstration", in HRI, 2007.
[14] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to Do and How to Do It: Translating Natural Language Directives into Temporal and Dynamic Logic Representation for Goal Management and Action Execution", in ICRA, 2009
[15] Ç Meriçli, S.D. Klee, J. Paparian, and M. Veloso, "An Interactive Approach for Situated Task Specification Through Verbal Instructions", in AAMAS, 2014.
[16] G. Gemignani, E. Bastianelli, and D. Nardi, "Teaching Robots Parametrized Executable Plans Through Spoken Interaction", in AAMAS, 2015.
[17] S. Mohan and J. E. Laird, "Learning Goal-Oriented Hierarchical Tasks from Situated Interactive Instruction", in AAAI, 2014.
[18] L. She, S. Yang, Y. Cheng, Y. Jia, J. Y. Chai, and N. Xi, "Teaching Robots New Actions through Natural Language Instructions", in Robot and Human Interactive Communication, 2014.
[19] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, and others, "PDDL-the planning domain definition language", 1998.
[20] T. Kollar, V. Perera, D. Nardi, and M. Veloso, "Learning Environmental Knowledge from Task-Based Human-Robot Dialog", in ICRA, 2013.