

# Housekeeping with Multiple Autonomous Robots: Knowledge Representation and Automated Reasoning for a Tightly Integrated Robot Control Architecture

Erdi Aker

Ahmetcan Erdogan

Esra Erdem

Volkan Patoglu

**Abstract**—We embed knowledge representation and automated reasoning in each level of the classical 3-layer robot control architecture, in such a way as to tightly integrate these layers. At the high-level, we represent not only actions and change but also commonsense knowledge in the action description language  $\mathcal{C}+$ . Geometric reasoning is lifted to the high-level by embedding motion planning in the domain description, using external predicates. Then a discrete plan is computed for each robot, using the causal reasoner CCALC. At the mid-level, if a continuous trajectory is not computed by a motion planner because the discrete plan is not feasible at the continuous-level, then formal queries are asked to the causal reasoner to find a different plan subject to some (temporal) conditions represented as formulas. At the low-level, if the plan execution fails, then a new continuous trajectory is computed by a motion planner at the mid-level or a new discrete plan is computed using an automated reasoner at the high-level. We apply this tightly integrated robot control architecture in a housekeeping domain with multiple autonomous robots, and illustrate this application with a simulation.

## I. INTRODUCTION

Consider a house consisting of three rooms: a bedroom, a living room and a kitchen as shown in Fig. 1. There are three cleaning robots in the house. The furniture is stationary and their locations are known to the robots a priori. Other objects are movable. There are three types of movable objects: books (green pentagon shaped objects), pillows (red triangular objects) and dishes (blue circular objects). The locations of some of the movable objects are not known to the robots a priori. Some objects are heavy and cannot be moved by one robot only; but the robots do not know which movable objects are heavy. The goal is for the cleaning robots to tidy the house collaboratively in a given amount of time.

This domain is challenging from various aspects:

- It requires representation of some commonsense knowledge. For instance, in a tidy house, books are in the bookcase, dirty dishes are in the dishwasher, pillows are in the closet. In that sense, books are expected to be in the living room, dishes in the kitchen and pillows in the bedroom. Representing such commonsense knowledge and integrating it with the action domain description (and the reasoner) is challenging.
- A robot is allowed to be at the same location with a movable object only if the object is being manipulated (attached, detached or carried); otherwise, robot-robot,

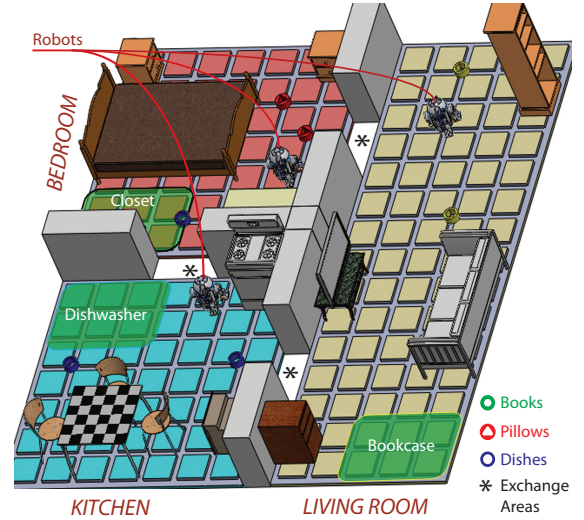


Fig. 1. Sample home simulation environment

robot-stationary object and robot-moveable object collisions are not permitted. Due to these constraints, representing preconditions of (discrete) actions that require (continuous) geometric reasoning for a collision-free execution is challenging. For instance, moving to some part of a room may not be possible for a robot because, although the goal position is clear, it is blocked by a table and a chair and the passage between the table and the chair is too narrow for the robot to pass through.

- When a plan execution fails, the robots may need to find another plan by taking into account some temporal constraints. For instance, when a robot cannot move an object because it is heavy, then the robots may want to compute another plan that postpones moving the heavy object to its goal position in the last four steps by the help of another robot. Representing such planning problems with temporal constraints and solving them are challenging.
- Solving the whole housekeeping problem may not be possible because the formalization gets too large for the reasoner. In that case, we can partition the housekeeping problem into smaller parts (e.g., each robot can tidy a room of the house). However, then the robots must communicate with each other to tidy the house collaboratively. For instance, if a robot cannot move a heavy object to its goal position, the robot may ask another robot for help. If the robot that cleans kitchen finds a book on the floor, then the robot should transfer it to the

robot that cleans the living room, by putting the book in the exchange area between kitchen and living room. Coordination of the robots in such cases, subject to the condition that the house be tidied in a given amount of time, is challenging.

- When a plan execution fails, depending on the cause of the failure, a recovery should be made. If a robot collides with a movable object whose presence and location is not known earlier (e.g., a human may bring the movable object into the room while the robot is executing the plan), then the robot may ask the motion planner to find a different trajectory to reach the next state, and continue with the plan. If the robot cannot find a trajectory, then a new plan can be computed to tidy the room. Monitoring executions of plans by multiple cleaning robots, taking into decisions for recovery from failures, is challenging.

We handle these challenges by representing the housekeeping domain in the action description language  $\mathcal{C}+$  [1] as a set of “causal laws” (Section III) and using the causal reasoner CCALC [2] for planning (Section IV), like in [3], [4], [5], [6], in the style of cognitive robotics [7]. We embed knowledge representation and automated reasoning in each level of the classical 3-layer robot control architecture, in such a way as to tightly integrate these layers. In particular, we make use of external predicates for the first two challenges. We represent commonsense knowledge as a logic program, and use the predicates defined in the logic program as external predicates in causal laws. Similarly, we implement collision checks as a function in the programming language C++, and use these functions as external functions in causal laws. In this way, we closely couple commonsense knowledge and geometric reasoning into the discrete reasoner (first layer). For the third challenge, we make use of queries. We represent planning problems as queries that allow us to add temporal constraints, and then we ask these queries to the causal reasoner to find new plans. Hence, high-level reasoning is involved in the motion planning (second layer). Finally, for the other challenges related to safe execution of plans (third layer), we introduce a planning and monitoring algorithm that solves the housekeeping problem by dividing it into smaller problems and then combining their solutions, that coordinates multiple cleaning robots for a common goal, and that ensures recovery from plan failures if possible (Section V).

## II. AN OVERVIEW OF ACTION LANGUAGE $\mathcal{C}+$

We describe action domains in the action description language  $\mathcal{C}+$ , by “causal laws.” Let us give a brief description of the syntax and the semantics of  $\mathcal{C}+$ ; we refer the reader to [1] for a comprehensive description.

We start with a (*multi-valued propositional*) *signature* that consists of a set  $\sigma$  of *constants* of two sorts, along with a nonempty finite set  $Dom(c)$  of *value names*, disjoint from  $\sigma$ , assigned to each constant  $c$ . An *atom* of  $\sigma$  is an expression of the form  $c = v$  (“the value of  $c$  is  $v$ ”) where  $c \in \sigma$  and  $v \in Dom(c)$ . A *formula* of  $\sigma$  is a propositional combination

of atoms. If  $c$  is a Boolean constant, we will use  $c$  (resp.  $\neg c$ ) as shorthand for the atom  $c = True$  (resp.  $c = False$ ).

A signature consists of two sorts of constants: *fluent constants* and *action constants*. Intuitively, fluent constants denote “fluents” characterizing a state; action constants denote “actions” characterizing an event leading from one state to another. A *fluent formula* is a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

An *action description* is a set of *causal laws* of three sorts. *Static laws* are of the form

$$\mathbf{caused} \ F \ \mathbf{if} \ G \tag{1}$$

where  $F$  and  $G$  are fluent formulas. *Action dynamic laws* are of the form (1) where  $F$  is an action formula and  $G$  is a formula. *Fluent dynamic laws* are of the form

$$\mathbf{caused} \ F \ \mathbf{if} \ G \ \mathbf{after} \ H \tag{2}$$

where  $F$  and  $G$  are as above, and  $H$  is a fluent formula. In (1) and (2) the part **if**  $G$  can be dropped if  $G$  is *True*.

The meaning of an action description can be represented by a “transition system”, which can be thought of as a labeled directed graph whose nodes correspond to states of the world and edges to transitions between states. Every state is represented by a vertex labeled with a function from fluent constants to their values. Every transition is a triple  $\langle s, A, s' \rangle$  that characterizes change from state  $s$  to state  $s'$  by execution of a set  $A$  of primitive actions.

While describing action domains, we can use some abbreviations. For instance, we can describe the (conditional) direct effects of actions using expressions of the form

$$c \ \mathbf{causes} \ F \ \mathbf{if} \ G \tag{3}$$

which abbreviates the fluent dynamic law

$$\mathbf{caused} \ F \ \mathbf{if} \ True \ \mathbf{after} \ c \wedge G$$

expressing that “executing  $c$  at a state where  $G$  holds, causes  $F$ .”

We can formalize that  $F$  is a precondition of  $c$  by the expression

$$\mathbf{nonexecutable} \ c \ \mathbf{if} \ \neg F \tag{4}$$

which stands for the fluent dynamic law

$$\mathbf{caused} \ False \ \mathbf{if} \ True \ \mathbf{after} \ c \wedge \neg F.$$

Similarly, we can prevent the execution of two actions  $c$  and  $c'$  by the expression

$$\mathbf{nonexecutable} \ c \wedge c'.$$

Similarly, we can express that  $F$  holds by default by the abbreviation

$$\mathbf{default} \ F.$$

We can represent that the value of a fluent  $F$  remains to be true unless it is caused to be false, by the abbreviation

$$\mathbf{inertial} \ F.$$

### III. REPRESENTATION OF HOUSEKEEPING DOMAIN

We represent the housekeeping domain in  $\mathcal{C}+$  and present it to CCALC for solving reasoning problems, such as planning. In the following, we will describe the housekeeping domain using the language of CCALC.

*a) Fluents and actions:* We view the house as a grid. The robots and the endpoints of objects are located at grid-points. We consider the fluents  $\text{at}(\text{TH}, X, Y)$  (“thing  $\text{TH}$  is at  $(X, Y)$ ”) and  $\text{connected}(\text{R}, \text{EP})$  (“robot  $\text{R}$  is connected to endpoint  $\text{EP}$ ”), and the actions  $\text{goto}(\text{R}, X, Y)$  (“robot  $\text{R}$  goes to  $(X, Y)$ ”),  $\text{detach}(\text{R})$  (“robot  $\text{R}$  detaches from the object it is connected to”), and  $\text{attach}(\text{R})$  (“robot  $\text{R}$  attaches to an object”). The fluents are inertial and the actions are exogenous. CCALC allows us to include this information at the very beginning of the action description while declaring fluent constants and action constants.

*b) Direct effects of actions:* We describe the direct effects of the actions above by causal laws of the form (3). For instance, the following causal law expresses the direct effect of the action of a robot  $\text{R}$  going to location  $(X, Y)$ :<sup>1</sup>

```
goto(R, X, Y) causes at(R, X, Y).
```

Similarly, we can describe the direct effects of the action of a robot  $\text{R}$  detaching from the endpoints of an object it is connected to:

```
detach(R) causes -connected(R, EP)
if connected(R, EP).
```

To describe the direct effects of the action of a robot  $\text{R}$  attaching to an endpoint of an object, we introduce an “attribute”  $\text{attach\_point}$  of this action to show at which endpoint the robot is attaching. Attributes of actions is a useful feature of CCALC that allows us to talk about various special cases of actions without having to modify the definitions of more general actions. We can formalize the direct effect of attaching a payload (“robot  $\text{R}$  is connected to the endpoint  $\text{EP}$  of an object”):

```
attach(R) causes connected(R, EP)
if attach_point(R)=EP.
```

*c) Preconditions of actions:* We describe effects of actions by causal laws of the form (4). For instance, we can describe that a robot  $\text{R}$  cannot go to a location  $(X, Y)$  if the robot is already at  $(X, Y)$ , by the causal laws:

```
nonexecutable goto(R, X, Y) if at(R, X, Y).
```

To describe that a robot  $\text{R}$  cannot go to a location  $(X, Y)$  if that location is already occupied by a stationary object, we need to know in advance the locations of stationary objects in the house. Such knowledge is represented as the “background knowledge” in Prolog. CCALC allows to use the predicates defined as part of background knowledge, in causal laws, as follows:

```
nonexecutable goto(R, X, Y) where occupied(X, Y).
```

<sup>1</sup>To present formulas to CCALC, conjunctions  $\wedge$ , disjunctions  $\vee$ , implications  $\supset$ , negations  $\neg$  are replaced with the symbols  $\&$ ,  $++$ ,  $->>$ , and  $-$  respectively.

where  $\text{occupied}(X, Y)$  describe the locations  $(X, Y)$  occupied by stationary objects.

In general, the *where* parts in causal laws presented to CCALC include formulas that consist of “external predicates/functions”. These predicates/functions are not part of the signature of the domain description (i.e., they are not declared as fluents or actions). They are implemented as functions in some programming language of the user’s choice, such as  $\mathcal{C}++$ . External predicates take as input not only some parameters from the action domain description (e.g., the locations of robots) but also detailed information that is not a part of the action domain description (e.g., geometric models). They are used to externally check some conditions under which the causal laws apply, or externally compute some value of a variable/fluent/action. For instance, suppose that the external predicate  $\text{path\_exists}(\text{R}, X, Y, X1, Y1)$  (implemented in  $\mathcal{C}++$ ) holds if there is a collision-free path between  $(X, Y)$  and  $(X1, Y1)$  for the robot  $\text{R}$ . Then we can express that the robot  $\text{R}$  cannot go from  $(X1, Y1)$  to  $(X, Y)$  where  $\text{path\_exists}(\text{R}, X, Y, X1, Y1)$  does not hold, by a causal law presented to CCALC as follows:

```
nonexecutable goto(R, X, Y) if at(R, X1, Y1)
where -path_exists(R, X1, Y1, X, Y).
```

Now consider the action of a robot  $\text{R}$  attaching to an endpoint of an object. This action is not possible if the robot is connected to some endpoint  $\text{EP}$  of an object:

```
nonexecutable attach(R) if connected(R, EP).
```

Note that here we do not refer to the special case of the action of attaching via attributes. Also it is not possible if the robot and the endpoint are not at the same location  $(X, Y)$ :

```
nonexecutable attach(R) & attach_point(R)=EP
if -[\X \Y | at(R, X, Y) & at(EP, X, Y)].
```

In the last line above, the negated expression stands for a disjunction of conjunctions  $\text{at}(\text{R}, X, Y) \& \text{at}(\text{EP}, X, Y)$  over locations  $(X, Y)$ .

Similarly, we can describe the preconditions of detaching from an object.

*d) Ramifications:* We describe two ramifications of the action of a robot  $\text{R}$  going to a location  $(X, Y)$ . If the robot is connected to an endpoint of an object, then the location of the object changes as well:

```
caused at(EP, X, Y)
if connected(R, EP) & at(R, X, Y).
```

Furthermore, neither the robot nor the endpoint are at their previous locations anymore:

```
caused -at(TH, X, Y) if at(TH, X1, Y1)
where X\=X1 ++ Y\=Y1.
```

Here  $\text{TH}$  denotes a “thing” which can be either a robot or an endpoint.

*e) Constraints:* We ensure that two objects do not reside at the same location by the constraint

```
caused false if at(EP, X, Y) & at(EP1, X, Y)
where EP \= EP1.
```

and that a robot is not connected to two endpoints by the constraint

```
caused false
  if connected(R,EP1) & connected(R,EP)
  where EP \= EP1.
```

We need to express that a robot cannot move to a location and attach to or detach from an endpoint of an object at the same time.

```
nonexecutable goto(R,X,Y) & attach(R).
nonexecutable goto(R,X,Y) & detach(R).
```

f) *Commonsense knowledge*: To clean a house, the robots should have an understanding of the following: tidying a house means that the objects are at their desired locations. For that, first we declare a “statically determined fluent” `at_desired_location(EP)` describing that the endpoint of an object is at its expected position in the house. A statically determined fluent is much like a “derived predicate”: it is defined in terms of other fluents. We define `at_desired_location(EP)` as follows:

```
caused at_desired_location(EP) if at(EP,X,Y)
  where in_place(EP,X,Y).
default -at_desired_location(EP).
```

The second causal law expresses that normally the movable objects in an untidy house are not at their desired locations. The first causal law formalizes that the endpoint `EP` of an object is at its desired location if it is at some “appropriate” position  $(X,Y)$  in the right room. Here `in_place/3` is defined externally.

After defining `at_desired_location/1`, we can introduce a “macro” to define `tidy`:

```
:- macros
  tidy -> [/\EP | at_desired_location(EP)].
```

Finally, the robots need to know that books are expected to be in the bookcase, dirty dishes in the dishwasher, and pillows in the closet. Moreover, a bookcase is expected to be in the living-room, dishwasher in the kitchen, and the closet in the bedroom. We describe such background knowledge externally as a Prolog program. For instance, the external predicate `in_place/3` is defined as follows:

```
in_place(EP,X,Y) :- belongs(EP,Obj),
  type_of(Obj,Type), el(Type,Room),
  area(Room,Xmin,Xmax,Ymin,Ymax),
  X>=Xmin, X<=Xmax, Y>=Ymin, Y<=Ymax.
```

Here `belongs(EP,OBJ)`, `type_of(OBJ,Type)` describes the type `Type` of an object `Obj` that the endpoint `EP` belongs to, and `el(Type,Room)` describes the expected room of an object of type `Type`. The rest of the body of the rule above checks that the endpoint’s location  $(X,Y)$  is a desired part of the room `Room`.

Note that the expected location of an object depends on where it is: for instance, the expected location of a book on the floor of a kitchen is the exchange area between kitchen and living room (so that the robot whose goal is to tidy the living room can pick it up and put it in the bookcase); on the

other hand, the expected location of a plate on the floor of kitchen is the dishwasher. Therefore, `area/5` describes either an exchange area (if the object does not belong to the room where it is at) or a deposit area (if the object belongs to the room where it is at). Note also that knowledge about specific objects in a room as well as specific deposit and exchange areas are not common knowledge to all robots; each robot has a different knowledge of objects in their room.

#### IV. REASONING ABOUT HOUSEKEEPING DOMAIN

Given the action domain description and the background and commonsense knowledge above, we can solve various reasoning tasks, such as planning, using `CCALC`. However, the overall planning problem for three cleaning robots may be too large (considering the size of the house, number of the objects, etc.). In such cases, we can divide the problem into three smaller planning problems, assigning each robot to tidy a room of the house in a given amount of time.

Consider the housekeeping domain described above. The goal is for the cleaning robots to tidy the house collaboratively in a given amount of time: Robot 1 is expected to tidy the living room, Robot 2 the bedroom, and Robot 3 the kitchen.

Suppose that the locations of some of the movable objects are known to the robots a priori, as shown in Table I. Robot 1 knows that there are two books, `comics1` and `novel1`, on the living room floor. Robot 2, on the other hand, knows that there are two pillows, `redpillow1` and `bluepillow1`, and a plate, `plate1`, on the bedroom floor. The robots also know where to collect the objects, as shown in Table I. For instance, Robot 2 knows that, in the bedroom, the closet occupies the rectangular area whose corners are at  $(5,0)$ ,  $(5,3)$ ,  $(7,0)$ ,  $(7,3)$ . Robot 2 also knows that the objects that do not belong to bedroom, such as `plate1` of type `dish`, should be deposited to the exchange area between bedroom and kitchen, that occupies the points  $(3,7)-(5,7)$ .

Planning problems for each robot are shown in Table II. For instance, in the living room, initially Robot 1 is at  $(3,2)$ , whereas the books `comics1` and `novel1` are located at  $(1,2)$  and  $(6,3)$ . The goal is to tidy the room and make sure that the robot is free (i.e., not attached to any objects), in at most  $k$  steps. Here `free` is a macro, like `tidy`. The planning problem for Robot 1 is presented to `CCALC` as a query as follows:

```
:- query
maxstep :: 0..k;
0: at(r1,3,2), at(comics1,1,2), at(novel1,6,3);
maxstep: tidy, free.
```

`CCALC` finds a shortest plan, Plan 1, for this problem:

```
0: goto(r1,6,3)
1: attach(r1,attach_point=novel1)
2: goto(r1,13,3)
3: detach(r1)
4: goto(r1,1,2)
5: attach(r1,attach_point=comics1)
6: goto(r1,13,2)
7: detach(r1)
```

TABLE I  
BACKGROUND KNOWLEDGE FOR EACH ROBOT.

|               | Robot 1 in Living Room                           | Robot 2 in Bedroom                                           | Robot 3 in Kitchen                                 |
|---------------|--------------------------------------------------|--------------------------------------------------------------|----------------------------------------------------|
| Object: Type  | comics1 : book<br>novell1 : book                 | redpillow1 : pillow<br>bluepillow1 : pillow<br>plate1 : dish | pan1 : dish<br>spoon1 : dish                       |
| Deposit Area  | (13, 15, 2, 5)                                   | (5, 7, 0, 3)                                                 | (0, 2, 0, 3)                                       |
| Exchange Area | kitchen: (11, 12, 0, 0)<br>bedroom: (0, 0, 3, 4) | kitchen: (3, 4, 7, 7)<br>living-room: (7, 7, 3, 4)           | bedroom: (0, 0, 3, 4)<br>living-room: (3, 4, 7, 7) |

TABLE II  
PLANNING PROBLEMS FOR EACH ROBOT.

|               | Robot 1 in Living Room                                 | Robot 2 in Bedroom                                                              | Robot 3 in Kitchen                                 |
|---------------|--------------------------------------------------------|---------------------------------------------------------------------------------|----------------------------------------------------|
| Initial State | at (r1, 3, 2) at (comics1, 1, 2)<br>at (novell1, 6, 3) | at (r2, 5, 6) at (redpillow1, 2, 5)<br>at (bluepillow1, 3, 6) at (plate1, 6, 3) | at (r3, 1, 5) at (pan1, 3, 1)<br>at (spoon1, 3, 6) |
| Goal          | tidy, free                                             | tidy, free                                                                      | tidy, free                                         |

TABLE III  
EXECUTION OF THE PLANS COMPUTED BY CCALC FOR EACH ROBOT. THE ROWS THAT ARE NOT LABELED BY A TIME STEP ARE NOT PART OF THESE PLANS, BUT ARE IMPLEMENTED AT THE LOW-LEVEL.

| Time Step | Robot 1 in Living Room                       | Robot 2 in Bedroom                                      | Robot 3 in Kitchen                        |
|-----------|----------------------------------------------|---------------------------------------------------------|-------------------------------------------|
| 1         | goto (r1, 6, 3)                              | goto (r2, 6, 3)                                         | goto (r3, 3, 1)                           |
| 2         | attach (r1, novell1)                         | attach (r2, plate1)                                     | attach (pan1) - FAILURE<br>(Heavy object) |
|           |                                              |                                                         | Re-task planning                          |
| 3         | goto (r1, 13, 3)                             | goto (r2, 7, 3)                                         | goto (r3, 3, 6)                           |
| 4         | detach (r1)                                  | detach (r2)                                             | attach (r3, spoon1)                       |
| 5         | goto (r1, 1, 2)                              | goto (r2, 3, 6) - FAILURE<br>(Unknown object)           | goto (r3, 0, 2)                           |
|           |                                              | Re-motion planning                                      |                                           |
| 6         | attach (r1, comics1)<br>Get ready to help r3 | attach (r2, bluepillow1)                                | detach (r3)                               |
| 7         | help r3                                      | goto (r2, 5, 2)                                         | goto (r3, 3, 1), goto (r1, 4, 1)          |
| 8         | help r3                                      | detach (r2)                                             | attach (r3, pan1), attach (r1, pan2)      |
| 9         | help r3                                      | goto (r2, 2, 5)                                         | goto (r3, 0, 1), goto (r1, 1, 1)          |
| 10        | help r3<br>Get ready to continue plan        | attach (r2, redpillow1)                                 | detach (r3), detach (r1)                  |
| 11        | goto (r1, 13, 2)                             | goto (r2, 7, 1)                                         | -                                         |
| 12        | detach (r1)                                  | detach (r2)                                             | -                                         |
|           |                                              | Final check for previously unknown or relocated objects |                                           |
| 13        | -                                            | goto (r2, 4, 5)                                         | -                                         |
| 14        | -                                            | attach (r2, yellowpillow1)                              | -                                         |
| 15        | -                                            | goto (r2, 6, 1)                                         | -                                         |
| 16        | -                                            | detach (r2)                                             | -                                         |

While CCALC computes such a plan, CCALC calls a motion planner (based on Rapidly exploring Random Trees (RRTs) [8]) to check the preconditions of the action of a robot going to a location without any collisions.

## V. MONITORING THE CLEANING ROBOTS

For each cleaning robot, a plan is computed and executed by the planning and monitoring algorithm introduced in [9]. According to this algorithm, first, each robot obtains the current state  $s$  of the world from the sensor information. After that, each robot computes a plan  $\mathcal{P}$  of length less than or equal to a given nonnegative integer  $k$ , from the observed state  $s$  to a goal state (that satisfies the given goal  $g$ ) in a world described by a given action domain description  $\mathcal{D}$ . Plans are computed using CCALC.

Once a plan  $\mathcal{P}$  is computed, each robot starts executing it. However, a plan execution may fail, for instance, due to the unpredictable interference of human. Human may relocate an object which is to be carried by robot, or bring

a new movable object to the room. Since the robots are unaware of these dynamic changes, they may collide with these obstacles. Furthermore, while most of the moveable objects are carried with only one robot, some of these objects are heavy and their manipulation requires two robots. The robots do not know in advance which objects are heavy, but discover a heavy object only when they attempt to move it.

When such incidents occur, robots can identify the cause of the failure and act accordingly. When a robot collides with an unknown movable object while following its planned trajectory for a transition  $\langle s, A, s' \rangle$  the robot tries to calculate a new, collision-free trajectory  $\pi$  to reach the next state  $s'$  from its current configuration. Such a trajectory  $\pi$  is computed by a motion planner based on RRTs. If no such trajectory is calculated by the motion planner, the robot goes to a safe state  $s$  (possibly the previous state) and asks CCALC to find a new plan  $\mathcal{P}$  to reach the goal from  $s$  taking into account the recently discovered moveable objects.

When a plan fails because a robot attempts to carry an

object which is relocated by another robot, the robot observes the world and asks CCALC to find a new plan  $\mathcal{P}$  to reach the goal from the current state  $s$  taking into account the recently relocated moveable objects.

When a plan fails because a robot attempts to manipulate a heavy object, the robot asks for assistance from other robots so that the heavy object can be carried to its destination. However, in order not to disturb the other robots while they are occupied with their own responsibilities, the call for help is delayed as much as possible. With the observation that the manipulation of the heavy object takes 4 steps (get to the heavy object, attach to it, carry it, detach from it), this is accomplished by asking CCALC to find a new plan  $\mathcal{P}$  that manipulates the heavy object within the last  $i = 4, 5, 6, \dots$  steps of the plan only. Once such a plan is computed, one of the robots who are willing to help gets prepared (e.g., detaches from the object it is carrying, if there is any) and goes to the room of the robot who requests help. (Currently, task allocation is done randomly.)

Note that if a human brings some objects into the room, and these objects are not in collision course with the robots, they cannot be discovered until the robot observes the world. Therefore, after the execution of the plans, a final observation instance is triggered comparing current state of the room with its goal state. If there are any discrepancies, then the robots ask for new plans to reach the goal.

In this algorithm plans are executed by as follows. For each transition  $\langle s, A, s' \rangle$  of the history of the plan, if  $A$  is a primitive action of the form  $\text{goto}(R, X, Y)$ , then a trajectory is computed. If  $A$  is a primitive action of some other form (e.g.,  $\text{attach}(R)$  or  $\text{detach}(R)$ ), then the action is executed. If  $A$  is a concurrent action of the form  $\{A_1, A_2\}$  to be executed by two robots, then an available robot is found and instructed about the actions to be executed concurrently.

Table III shows the execution of plans by Robots 1–3. Robot 1 executes Plan 1 safely without any collisions, but goes to kitchen at time step 7 to help Robot 3 to move a heavy object to its goal position. Robot 2 starts executing its plan, but fails at time step 5 due to a collision with a pillow `bluePillow1` unknown to Robot 2 a priori. Fortunately, the motion planner finds another trajectory to reach the next state and Robot 2 continues with the execution of the rest of the plan after following the trajectory. After the completion of the plan at time step 13, Robot 2 checks whether there is any other pillow to be put into the closet. After discovering another pillow `yellowPillow1`, Robot 2 asks for a new plan to move it to the closet, and safely executes that plan as well. Robot 3 on the other hand starts executing a plan, but at time step 2, finds out that the pan `pan1` he wants to move is too heavy. Then Robot 3 goes to a safe state and asks for help to carry the heavy object to its goal position.

We showed the execution of these plans with a simulation, where the housekeeping domain is modeled using VRML 2.0 and the execution of the actions is implemented in Matlab/Simulink. Video clips illustrating this simulation can be found at <http://krr.sabanciuniv.edu/cogrobo/demos/housekeeping/>.

## VI. CONCLUSION

We formalized a housekeeping domain with multiple cleaning robots, in the action description language  $\mathcal{C}+$ , and solved some housekeeping problem instances using the reasoner CCALC as part of a planning and monitoring framework.

In this framework, we embedded knowledge representation and automated reasoning in each level of the classical 3-layer robot control architecture, in such a way as to tightly integrate these layers, using various utilities of CCALC. At the high-level, external predicates are used to embed geometric reasoning in causal laws, and attributes of actions are used to talk about the special cases of actions. To represent commonsense knowledge and background knowledge, we made use of external predicates/functions and macros in causal laws. At the mid-level, if a continuous trajectory is not computed by a motion planner because the discrete plan is not feasible at the continuous-level, then queries are used to find a different plan taking into account temporal constraints. At the low-level, if plan execution fails due to a collision with movable objects whose presence and location are not known in advance or due to heavy objects that cannot be lifted alone, the cause of the failure is identified and a recovery is found accordingly either by calling the motion planner to find a different continuous trajectory or by asking the causal reasoner queries to find a different plan. In this way, high-level knowledge representation and automated reasoning are utilized at each level.

A related work on the use of AI planning for housekeeping with multiple robots is [10], which considers a domain with no concurrency, no obstacles and no human intervention. On the other hand, it considers heterogenous robots with different functionalities. Application of our approach to domains with such robots is part of our ongoing work.

## REFERENCES

- [1] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, "Nonmonotonic causal theories," *Artificial Intelligence*, vol. 153, pp. 49–104, 2004.
- [2] N. McCain and H. Turner, "Causal theories of action and change," in *Proc. of AAAI/IAAI*, 1997, pp. 460–465.
- [3] O. Caldiran, K. Haspalamutgil, A. Ok, C. Palaz, E. Erdem, and V. Patoglu, "Bridging the gap between high-level reasoning and low-level control," in *Proc. of LPNMR*, 2009.
- [4] —, "From discrete task plans to continuous trajectories," in *Proc. of BTAMP*, 2009.
- [5] K. Haspalamutgil, C. Palaz, T. Uras, E. Erdem, and V. Patoglu, "A tight integration of task and motion planning in an execution monitoring framework," in *Proc. of BTAMP*, 2010.
- [6] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Proc. of ICRA*, 2011.
- [7] H. Levesque and G. Lakemeyer, "Cognitive robotics," in *Handbook of Knowledge Representation*. Elsevier, 2007.
- [8] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.
- [9] E. Aker, A. Erdogan, E. Erdem, and V. Patoglu, "Housekeeping with multiple autonomous robots: Representation, reasoning and execution," in *Proc. of Commonsense'11*, 2011.
- [10] R. Lundh, L. Karlsson, and A. Saffiotti, "Autonomous functional configuration of a network robot system," *Robotics and Autonomous Systems*, vol. 56, no. 10, pp. 819–830, 2008.