Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation

Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, Tansel Uras
Faculty of Engineering and Natural Sciences
Sabancı University, İstanbul, Turkey

***Abstract*— We present a formal framework that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning. The framework features bilateral interaction between task and motion planning, and embeds geometric reasoning in causal reasoning, thanks to several advantages inherited from its underlying components. In particular, our choice of using a causality-based high-level formalism for describing action domains allows us to represent ramifications and state/transition constraints, and embed in such formal domain descriptions externally defined functions implemented in some programming language (e.g., C++). Moreover, given such a domain description, the causal reasoner based on this formalism (i.e., the Causal Calculator) allows us to compute optimal solutions (e.g., shortest plans) for elaborate planning/prediction problems with temporal constraints. Utilizing these features of high-level representation and reasoning, we can combine causal reasoning, motion planning and geometric planning to find feasible kinematic solutions to task-level problems. In our framework, the causal reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic solution for that task-plan then the motion planner guides the causal reasoner by modifying the planning problem with new temporal constraints. Furthermore, while computing a task-plan, the causal reasoner takes into account geometric models and kinematic relations by means of external predicates implemented for geometric reasoning (e.g., to check some collisions); in that sense the geometric reasoner guides the causal reasoner to find feasible kinematic solutions. We illustrate an application of this framework to robotic manipulation, with two pantograph robots on a complex assembly task that requires concurrent execution of actions. A short video of this application accompanies the paper.**

I. Introduction

Manipulation planning aims automatic generation of robot motion sequences for manipulation of movable objects among obstacles to achieve a desired goal configuration. These problems involve objects that can only move when picked up by robots and the order of pick-and-place operations for manipulation may matter to obtain a feasible kinematic solution [15]. Therefore, geometric reasoning and motion planning alone are not sufficient to solve them; and planning of actions such as the pick-and-place operations need to be integrated with the motion planning problem.

The state-of-the-art motion planning systems have been extended to handle some manipulation planning problems [14], [1] based on the idea of viewing the configuration space as consisting of regions connected by lower-dimensional submanifolds, such as transit and transfer manifolds. However, such manipulation planners are domain-specific and cannot handle action planning in a generic manner: each one addresses a specific sort of manipulation planning problems without making use of task planning. As a result, there has been a growing interest in hybrid manipulation planning approaches that utilize both task planning and motion planning. The traditional approaches to hybrid manipulation planning have been top-down, separating high-level task planning from lower-level motion planning. In these approaches, task planning is simplified by ignoring low-level (geometric) details; however, due to such an abstraction, the resulting plans may be inefficient/infeasible.

Recently, more elaborate approaches have been proposed to integrate task and motion planning more tightly *at the search level*. For instance, [8], [10], [12], [18], [19] take advantage of a forward-search task planner to incrementally build a task plan, while checking its kinematic/geometric feasibility at each step by a motion planner; all these approaches use different methods to utilize the information from the task-level to guide and narrow the search in the configuration space. By this way, the task planner helps focus the search process during motion planning. However, each one of these approaches presents a specialized combination of task and motion planning at the search level, and do not consider a general interface between task and motion planning.

With this motivation, [3], [5] introduce an alternative approach to integrating task and motion planning by considering a general interface between them, using "external predicates/functions" whose values for ground instances are computed by an external mechanism, e.g., by a C++ program. The concept of external predicates/functions is not new; they have existed as undocumented features of the planner TLPlan [2] and the Causal Calculator (CCALC) [17]. The idea in [3], [5] is to use external predicates/functions in the action domain description, for checking the feasibility of a primitive action by a motion planner. [3] applies this approach in the action description language $\mathcal{C}+$ [7] using CCALC, while [5] extends the planning domain description language PDDL [6] to support external predicates/functions (called semantic attachments) and modifies the planner FF [11] accordingly.

Our contribution is an alternative approach for integrating task and motion planners that combines various advantages of some of the related approaches discussed above with some other advantages inherited from the high-level causality-based representation and reasoning formal framework we use in our studies. As in [3], [5], we also consider a

general interface between high-level causal reasoning and low-level geometric reasoning and motion planning, using external predicates/functions, but in a more flexible way. The first novelty of our approach is the flexible use of external predicates/functions for feasibility checks: Instead of delegating all sorts of feasibility checks to external predicates as in [3], [5], we can decide which sorts of feasibility check should be done by external predicates as part of high-level reasoning, and which sorts of feasibility checks should be done by motion planning later on. For instance, external predicates can check only collisions of robots with each other and with other objects (so they do not check, for instance, collisions of objects with each other), and they can be used in action domain description to specify conditional effects of these robots' actions. By this way, geometric reasoning is "embedded" in high-level representation: while computing a task-plan, the causal reasoner takes into account geometric models and kinematic relations by means of external predicates implemented for geometric reasoning. In that sense, the geometric reasoner guides the causal reasoner to find feasible kinematic solutions. Such a flexibility may be useful if gathering all sorts of feasibility checks in one external predicate/function is computationally disadvantageous, or if checking feasibility in a modular way (using different mechanisms/algorithms/solvers) is preferred.

Since we do not delegate all the feasibility checks to external predicates/functions, in addition to the bilateral interactions between task and motion planing as discussed above (i.e., task plans guide search in motion planning, motion planners check feasibility of task plans and thus affect search of task planners), we need to find tighter interactions between task and motion planning to handle all feasibility checks for a guaranteed-feasible kinematic solution. This requirement brings out the second novelty of our approach — another interaction between causal reasoning and motion planning: when a motion planning failure occurs due to some infeasibility not captured by geometric reasoning handled by external predicates/functions, the description of the planning problem is modified taking into account some domain-specific information from motion-level (e.g., the causes of infeasibilities), and the causal reasoner is asked to solve a more "relevant" planning problem. Therefore, instead of guiding the task planner at the *search level* by manipulating its search algorithm directly, the motion planner guides the task planner at the *representation level* by presenting to it the "right" planning problem.

Compared with the recent efforts for integration of task and motion planning at the search level, bringing (a certain amount of) geometric/kinematic details to the representation level allows one to utilize different formulations (declarative or procedural) and reasoning systems. Other novelties of our approach are of this sort: they are inherited from the high-level causality-based representation and reasoning formalism (action language $\mathcal{C}+$) and its automated reasoner (CCALC) we use in our studies. Due to the expressiveness of the formalism, we can handle concurrent actions by multiple agents, nondeterministic effects, multi-valued actions/fluents,

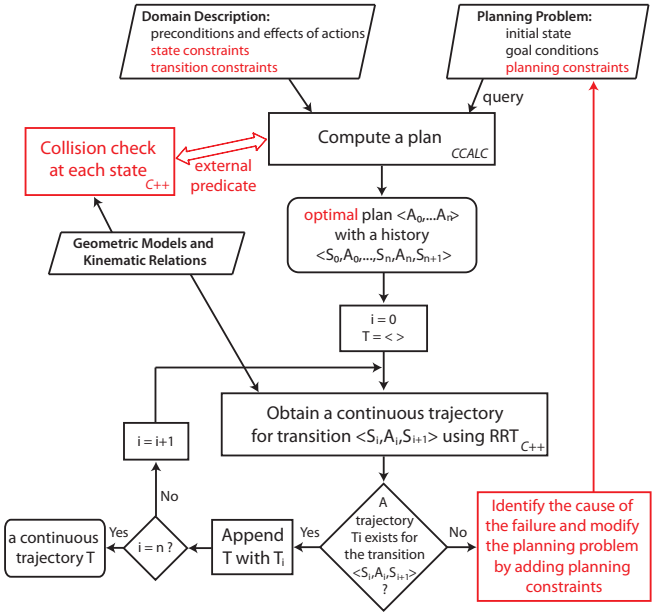

Fig. 1. The overall system architecture. The components depicted in red signify the important aspects of our approach.

additive fluents for reasoning about shared resources, state/-transition constraints, and change that do not involve actions (e.g., ramifications of actions), defaults, etc. Due to the input language of the causal reasoner, we can use external predicates/functions implemented in some programming language of the users' choice, and we can solve modified planning problems that involve temporal constraints. Due to the search mechanism of the causal reasoner, we can compute optimal plans (e.g., in terms of its length or total cost of actions). Due to modular structure of the causal reasoner, we can experiment with various search engines, such as (parallel) SAT solvers, without modifying their algorithms, and thus inherit advantages of these underlying solvers. Due to the capabilities of the causal reasoner, our approach substantially extends the classes of manipulation problems that can be solved. In particular, we can solve not only planning but also prediction/postdiction and diagnosis problems.

In the following, instead of describing all these inherited novelties, we emphasize the tight interactions between causal reasoning, geometric reasoning, and motion planning.

II. The Overall System Architecture

Our aim eventually is to compute a complete continuous trajectory for each robot to reach a common goal in an optimal way, considering the possibility of concurrent executions of actions by multiple robots. Such a problem cannot be solved, in general, by either motion planning (because the order of actions matter, in particular with picks/drops) or task planning (because the problem is too large and detailed). We solve this problem by dividing the problem into smaller parts and by making use of computational methods for both sorts of problems in such a way that they guide each other. The overall architecture of our formal framework

that combines high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning is illustrated in Fig. 1.

We start with an action domain description and a planning problem description in the input language of CCALC, geometric models in VRML, and kinematic relations as a C++ program.

- A description of geometric models include specifications of the geometry of robots, payloads, and other objects (e.g., obstacles) in the environment.
- Kinematic relations between robot end-effector configurations and robot joint configurations are implemented as functions in C++.
- An action domain description is a set of "causal laws" (causality-based formulas) that express preconditions and direct effects of actions of robots, causal relations that do not involve these actions directly (e.g., ramifications), and state and transition constraints. These causal laws may include "external predicates" to express conditions that involve geometric reasoning. For instance, a precondition of an action such as "moving a payload" may be that the robot does not collide with other robots. Such a collision check involves geometric reasoning, and can be implemented as a function in some programming language, say C++, making use of the given geometric models and kinematic relations. This function can be included in the causal law expressing the precondition of "moving a payload" as an external predicate, so that geometric models are also taken into account while a task plan is computed.
- A planning problem description is a set of formulas that express an initial state (or a set of initial states, in case of uncertainty), goal conditions, and temporal constraints.

Given an action domain description and a planning problem, first we compute an optimal plan (a sequence of actions) $\langle A_0, \ldots, A_n \rangle$ and its complete history (including intermediate states) $\langle S_0, A_0, S_1, \ldots, S_n, A_n, S_{n+1} \rangle$ using CCALC. The computed plan may involve concurrent execution of actions by multiple robots; so each A_i is a set of primitive actions. The optimality of a plan can be defined in terms of its length (the value of n) or the total cost of actions; in the following, we consider the former.

Next, given a discrete plan and its history, and considering the given geometric models and kinematic relations, a collision-free trajectory T for each robot (if one exists) is computed by our motion planner, based on Rapidly exploring Random Trees (RRT) [16]. Initially T is empty. For each transition $\langle S_i, A_i, S_{i+1} \rangle$ in the given history, the motion planner tries to compute a continuous trajectory T_i. If such a transition T_i is found then it is appended to the end of T.

If the motion planner fails to find a continuous trajectory for a transition, we identify the cause of that failure. For instance, consider the transition $\langle S_i, A_i, S_{i+1} \rangle$ that describes "two robots moving a payload from one location to another, each robot holding from one end-point of the payload". Suppose that it is not possible to place the payload at the destination location because some part of it collides with some obstacle at state S_{i+1}. Then the motion planner cannot find a trajectory for this transition. In that case, the cause of the failure can be characterized by the state S_{i+1}. Such a failure can be avoided by modifying the planning problem by adding a constraint that expresses "S_{i+1} should not be possible". Alternatively, such a transition may not be possible because the payload collides with some obstacle while moving from one location to another. In that case, the cause of the failure can be characterized by the state S_i and the action A_i. Such a failure can be avoided by modifying the planning problem by adding a temporal constraint that expresses "A_i should not be executed at S_i at any time". After that, the modified planning problem is solved by CCALC, generating a different optimal task plan.

It is important to emphasize here two types of relations between different kinds of problem solving. First, the bilateral interaction between causality-based reasoning and motion planning: the causal reasoner guides the motion planner by finding an optimal task-plan; if there is no feasible kinematic solution for that task-plan then the motion planner guides the causal reasoner by modifying the planning problem with new temporal constraints. Second, the embedding of geometric reasoning in causal reasoning: while computing a task-plan, the causal reasoner takes into account geometric models and kinematic relations by means of external predicates implemented for geometric reasoning (e.g., to check some collisions); in that sense the geometric reasoner guides the causal reasoner to find feasible kinematic solutions. In the following, we illustrate these two aspects of our approach in more detail with an example.

III. Example: Two Robots and Multiple Payloads

Consider two robots and multiple payloads on a platform. The payloads can be manipulated by the end effectors of the robots. In particular, the end-effector of each robot can pick (hold and elevate) or drop (release) the payload at one of its end points. None of the robots can carry the payload alone; they have to pick the payload at both ends. Since the payload is elevated from the platform when the robots are holding it, the payload can not collide with the other payloads. However, collisions between payloads may occur if a payload is dropped on top of another one and such collisions are not permitted. Similarly, other types collisions (robot-robot, payload-obstacle and robot-obstacle) are not permitted either.

Initially, a configuration of the payloads on the platform is given (e.g., as in Fig. 2(a)). The goal is to reconfigure the payloads in an optimal manner (by minimum number of steps). This problem requires payloads to be picked and placed a number of times before they can be positioned into their final configuration. Due to the constraint that a payload can be carried by two robots only and due to the optimality of the plan, this problem requires concurrency. Another challenge, meanwhile, is to avoid collisions of the payloads with each other.

IV. Embedding Geometric Reasoning in Action Domain Descriptions

Let us first describe the action domain of the example above, in the input language of CCALC (i.e., the action description language $\mathcal{C}+$).

We view the platform as a grid. We represent the robots by the constants `r1` and `r2`. We denote the payloads by nonnegative integers, and denote the endpoints of a payload i by the nonnegative integers $2i$ and $2i-1$.

We characterize each robot by its end-effector, and describe its position by a grid point: the location `(X,Y)` of a robot `R` is specified by two functional fluents, `xpos(R)=X` and `ypos(R)=Y`. Similarly, the location `(X,Y)` of an end point `P1` of the payload is specified by two fluents, `xpay(P1)=X` and `ypay(P1)=Y`. Movements of a robot `R` in some direction `D` are described by actions of the form `move(R,D)`. Each such action has an attribute that specifies the number of steps to be taken by the robot. In addition, we denote the actions of picking and dropping a payload by `pick(R)` and `drop(R)`; the former action has an attribute that specifies at which endpoint the robot picks the payload.

In the following, suppose that `R` denotes a robot, `P1` and `P2` denote the end points of a payload, `N` and `N1` range over nonnegative integers `1`, ..., `maxN`, and `D` and `D1` range over all directions, `up`, `down`, `right`, `left`. Also suppose that `X1`, `X2`, `Y1`, `Y2` range over nonnegative integers `1`, ..., `maxXY`.

A. Representing Actions and Change

We describe the preconditions and the effects of the actions as in [3]. For instance, we describe the direct effect of a robot's picking a payload, by the causal laws

```
pick(R) causes holding(R,P) if pickpoint(R)=P.
```

These causal laws express that, after a robot `R` picks a payload at its endpoint `P`, the robot is holding it at its endpoint `P`.

One of the preconditions of the action of a robot `R`'s picking a payload at its endpoint `P` is that "`R` should not be holding `P`"; otherwise, the action is not executable. This is expressed by the causal laws

```
nonexecutable pick(R) if holding(R,P).
```

We can also express conditions on the concurrent executability of actions. For instance, two robots `R` and `R1` cannot pick a payload at the same endpoint:

```
nonexecutable pick(R) & pick(R1)
  if pickpoint(R)=pickpoint(R1) & R@<R1.
```

A robot `R` cannot pick a payload while moving:

```
nonexecutable move(R,D) & pick(R).
```

In addition to causal relations that involve actions, as in the preconditions and direct effects of actions above, we can also express causal relations that do not involve actions directly. For instance, if a robot `R` is holding a payload `P1`, then the location of the payload is the same as the location `(X1,Y1)` of the robot.

```
caused xpay(P1)=X1 if holding(R,P1) & xpos(R)=X1.
caused ypay(P1)=Y1 if holding(R,P1) & ypos(R)=Y1.
```

Such causal laws allow us to reason about ramifications of actions without describing them: whenever a robot moves then the payload it holds moves as its indirect effect.

Finally, we can add state/transition constraints to ensure some conditions. For instance, we can prohibit states where the robots hold different payloads:

```
caused false if holding(R1,P1) &
   holding(R2,P2) & R1@<R2 & P1\=P2
   where -samePayload(P1,P2).
```

where `samePayload(P1,P2)` describes that the endpoints `P1` and `P2` belong to the same payload. Such causal laws allow us to reason about qualifications of actions without describing them: the robots cannot pick different payloads.

B. Embedding Geometric Reasoning in Causal Laws

We can embed geometric reasoning in such an action domain description in two ways: using state constraints and using external predicates. Let us first consider collisions of payloads with each other. We can identify the conditions under which payloads collide with each other, provided that the orientations and the lengths of the payloads, as well as the positions of their leftmost bottom endpoints are given. For instance, consider two payloads `K1` and `K2` of length `lengthP` on the board, whose orientations are vertical. Suppose that the left bottom endpoints of these payloads are `(minXP(K1),minYP(K1))` and `(minXP(K2),minYP(K2))`. Then these payloads collide with each other if `abs(minYP(K1)-minYP(K2))=<lengthP`. Once such collision conditions are identified, we can prevent them:

```
caused false if orientationP(K1)=v &
  orientationP(K2)=v & K1@<K2 & -beingCarried(K1) &
  -beingCarried(K2) & minXP(K1)=minXP(K2) &
  abs(minYP(K1)-minYP(K2))<lengthP.
```

Similarly, we can identify conditions for collisions between payloads with different orientations, and add causal laws to prevent such cases.

Next let us consider collisions between the robots, or between a robot and obstacles. To detect this sort of collisions, we need to know the geometric models and kinematic relations; however, such detailed information is not represented at the high-level (otherwise, if we could represent it, the domain description and thus the planning problem would be too large for the causal reasoner). Fortunately, CCALC supports "external predicates", which are functions implemented in some programming language (e.g., C++).

An external predicate takes 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 not the action domain description (e.g., geometric models); it returns a truth value. For instance, we check whether a robot located at `(X1,Y1)` collides with another robot at `(X2,Y2)`, by an external predicate `collision(X1,Y1,X2,Y2)` implemented as a C++ program. Then we can add causal laws to ensure that the robots do not collide with each other:

```
caused false if xpos(r1)=X1 & ypos(r1)=Y1 &
  xpos(r2)=X2 & ypos(r2)=Y2
  where collision(X1,Y1,X2,Y2).
```

In addition, an external predicate can accomplish some other tasks as "side-effects". For instance, while checking whether a robot located at `(X1,Y1)` collides with another robot at `(X2,Y2)`, the external predicate `collision(X1,Y1,X2,Y2)` can form a database keeping which locations lead to a collision and which locations do not. Then this database can be reused in the future.

V. Bilateral Interaction between Causal Reasoning and Motion Planning

With the action domain description and the external predicate above, CCALC combines causal reasoning with geometric reasoning to compute task plans without robot-robot or robot-obstacle collisions. For instance, consider the environment in Fig. 2. Suppose that initially the robots `r1` and `r2` are at `(1,1)` and `(9,9)` respectively; the first payload is located at `(3,2)` and `(8,2)`; the second payload is located at `(8,5)` and `(3,5)`; the third payload is located at `(9,3)` and `(9,8)`. The goal is to move the payloads to the following locations: first payload to `(3,2)` and `(3,7)`; the second payload to `(6,7)` and `(6,2)`; the third payload to `(9,8)` and `(9,3)`. This planning problem can be described in the language of CCALC by means of a "query" as follows:

```
:- query  % Query 1
maxstep:: 0 ..infinity;
0: % Initial state
 % robot 1
 xpos(r1)=1, ypos(r1)=1,
 % robot 2
 xpos(r2)=9, ypos(r2)=9,
 % endpoints (1 and 2) of payload 1
 xpay(1)=3, ypay(1)=2, xpay(2)=8, ypay(2)=2,
 % endpoints (3 and 4) of payload 2
 xpay(3)=8, ypay(3)=5, xpay(4)=3, ypay(4)=5,
 % endpoints (5 and 6) of payload 3
 xpay(5)=9, ypay(5)=3, xpay(6)=9, ypay(6)=8;
maxstep: % Goal conditions
 % endpoints of payload 1
 xpay(1)=3, ypay(1)=2, xpay(2)=3, ypay(2)=7,
 % endpoints of payload 2
 xpay(3)=6, ypay(3)=7, xpay(4)=6, ypay(4)=2,
 % endpoints of payload 3
 xpay(5)=9, ypay(5)=8, xpay(6)=9, ypay(6)=3,
 % robots must not be holding any payloads
 [/\R /\P | -holding(R,P)].
```

This query, Query 1, asks for a plan with the minimum number of time steps. CCALC then computes the following plan (Plan 1) of length 27 for this problem:

```
0:  move(r1, up, steps=3) move(r1, left, steps=1)
    move(r2, down, steps=4)
...
13: pick(r1, pickpoint=1) pick(r2, pickpoint=2)
14: move(r1, down, steps=2) move(r1, right, steps=2)
    move(r2, up, steps=3) move(r2, left, steps=2)
...
26: drop(r1) drop(r2)
```

Note that each step of the plan involves concurrent execution of a set of primitive actions. For instance, at time step 13, both robots pick the opposite endpoints `1` and `2` of the payload `1` at the same time. At time step 14, the robot `r1` moves down by two units and right by two units at the same time (note that this concurrent action essentially describes a diagonal move of the robot); meanwhile, the robot `r2` moves up by three units and left by two units.

A. Task Planning guides Motion Planning

Our aim eventually is to compute a complete continuous trajectory for each robot to reach a common goal in an optimal way, considering the possibility of concurrent executions of actions by multiple robots. Such a problem cannot be solved, in general, directly by motion planning because the order of actions matter, in particular with picks/drops. It can be solved, on the other hand, by the help of the causal reasoner: first a task plan and its history is computed, and the motion planner is called to find a continuous collision-free trajectory for each transition. This is described more precisely in Algorithm 1, implemented in Python.

Given a discrete plan and its history $\langle S_i, A_i, S_{i+1}\rangle$, and considering the given geometric models and kinematic relations, a collision-free trajectory Π for all robots (if one exists) is computed by a motion planner. Initially T is empty. For each transition $\langle S_i, A_i, S_{i+1}\rangle$ $(i = 0, 1, ..., n)$ in the given history, the motion planner tries to compute a continuous trajectory T_i. If such a transition T_i is found then it is appended to the end of T. Otherwise, the problem is infeasible; in such a case, the motion planner can ask for a different task plan as explained in the next subsection.

A number of motion planning algorithms are suitable for this framework. For instance, Probabilistic Road Maps (PRM) [13] could be considered a good candidate for smaller number of possible states for utilizing their multiple query

Algorithm 1 TASK&MOTION_PLAN

Input: Action domain description $\mathcal{D}$, and planning problem $\mathcal{P}$ with the initial state(s) S and the goal G
while *true* **do**
  plan, $P \leftarrow$ Compute a shortest task plan P of length n (within a history $H = \langle S_0, A_0, S_1, ..., S_n, A_n, S_{n+1}\rangle$, where $S_0 = S$ and $S_{n+1} = G$) using CCALC with $\mathcal{D}$ and $\mathcal{P}$ (if there is such a plan);
  if $\neg$*plan* **then**
    return *false*;
  end if
  $T := \langle\rangle$; // Initially the trajectory is empty
  trajectoryFound := *true*;
  $i := 0$;
  while *trajectoryFound* **do**
    $\langle S_i, A_i, S_{i+1}\rangle \leftarrow$ Extract from H the next transition;
    // Compute a trajectory π for $\langle S_i, A_i, S_{i+1}\rangle$, if one exists
    trajectoryFound, $\pi \leftarrow$ MOTION_PLAN(S_i,S_{i+1});
    if $\neg$*trajectoryFound* **then**
      $\mathcal{P} \leftarrow$ Identify the cause of the failure and modify the planning problem $\mathcal{P}$ accordingly;
    else
      $T \leftarrow$ Append π to T;
      if A_i is the last action **then**
        return *true*, P, H, T;
      end if
    end if
    i++;
  end while
end while

nature, i.e., ability to use a PRM multiple times. For our example problem, however, the number of possible states increases exponentially with the increase in grid resolution, and this makes enumeration and initial sampling a problem. In addition, sampling the entire work space is not efficient. With these considerations, we use a single-query method that quickly responds: a greedy variation of RRTs.

B. Motion Planning guides Task Planning

If the motion planner fails to find a continuous collision-free trajectory for a transition $\langle S_i, A_i, S_{i+1} \rangle$, first we try to identify the cause of that failure. In our example, such a failure occurs in three cases: the given time/sample threshold is too low, the payload collides with an obstacle during the transition from S_i to S_{i+1} (exclusive) while being carried, or the payload collides with an obstacle at state S_{i+1}. In the first case, motion planning can be modified by increasing the threshold. In the first two cases, task planning can be modified as follows. The cause of the failure can be characterized by the state S_i and the action A_i; and such a failure can be avoided by modifying the planning problem by adding a temporal constraint that expresses "A_i should not be executed at S_i". In the last case, the cause of the failure can be characterized by the state S_{i+1}. Such a failure can be avoided by modifying the planning problem by adding a constraint that expresses "S_{i+1} should not be possible". After that, the modified planning problem is solved by CCALC, generating a different optimal task plan that does not cause such failures.

For instance, consider the transition $\langle S_3, A_3, S_4 \rangle$ from the history of Plan 1:

```
3: holding(r1,4), holding(r2,3),
   xpay(3)=8, ypay(3)=5, xpay(4)=3, ypay(3)=5,...,
   move(r1, down, steps=3) move(r1, right, steps=1);
4: xpay(3)=8, ypay(3)=5, xpay(4)=4, ypay(4)=2,...;
```

where the end-points of the second payload (that the robots are holding) are at `(4,2)` and `(8,5)`. The motion planner cannot find a continuous trajectory for this transition since the payload collides with an obstacle at Step 4 (third kind of failure). Then, Query 1 is modified by adding a constraint as follows:

```
:- query  % Query 2
maxstep :: 0..infinity;
0: ...; % Initial states
maxstep: ...; % Goal conditions
% Constraints
T<maxstep ->>
  -((T: xpay(3)=8) && (T: xpay(4)=4) &&
    (T: ypay(3)=5) && (T: ypay(4)=2) && ...).
```

to ensure that CCALC does not consider S_4 as a possible state. Then CCALC computes a different plan (Plan 2) without such a failure.

```
0:  move(r1, up, steps=4)    move(r2, left, steps=1)
...
13: pick(r1, pickpoint=1) pick(r2, pickpoint=2)
14: move(r1, down, steps=1) move(r2, up, steps=3)
    move(r2, left, steps=2)
...
26: drop(r1) drop(r2)
```

Thus, for each action of this plan, the motion planner can find a continuous collision-free trajectory (Fig. 2).

VI. Execution and Results

We have tested the applicability and effectiveness of our framework using two Pantograph robots (two degrees-of-freedom planar parallel manipulators) to perform a complex assembly task that requires concurrent execution of actions. In particular, we used symmetric wooden sticks with metal tips as payloads. To enable pick and drop actions, we equipped the end-effectors of the Pantograph robots with linear servo motors with magnetic tips acting out of plane. The magnetic tip of the linear servo motors can pick (hold and elevate) a payload at one of its end points. Similarly, the payload can be dropped by retracting the magnetic tip inside its case. The robots were closed loop controlled at 100 Hz to ensure robust trajectory tracking of their end-effectors. The controllers of the robots have been implemented on a PC-based control architecture, that compromises of a PCI I/O card and a workstation, simultaneously running RTX real-time operating system and Windows XP SP2.

Execution of Plan 2 on the physical robot setup can be viewed using the accompanying video file. Fig. 2 depicts the experimentally recorded trajectories of the robots during this plan execution. In this figure, the first six steps of the plan are shown in Fig. 2(b). Observe that the successful completion of this plan necessitates payloads to be picked and dropped a number of times before they can be arranged to their final configuration. For instance, the robots first pick Payload 2 and drop it to an intermediate location (Fig. 2(b)), then move Payload 3 to an intermediate location (Fig. 2(c)), then place Payloads 1 and 3 into their goal positions (Fig. 2(d) and (e)), and finally move Payload 2 to its final position (Fig. 2(f)).

VII. Conclusion

We have presented an alternative approach to combine high-level representation and causality-based reasoning with low-level geometric reasoning and motion planning. The main contributions of this approach are: 1) flexible use of external predicates/functions for some feasibility checks allows a generic interface between high-level causal reasoning and low-level geometric reasoning and motion planning; 2) embedding of geometric reasoning in high-level representation allows the causal reasoner to take into account geometric models and kinematic relations; 3) causal reasoning guides motion planning by presenting the order of actions; 4) motion planning guides causal reasoning at the *representation level* by modifying description of the planning problem to take into account some domain-specific information such as the causes of infeasibilities, and asking the causal reasoner to solve a more "relevant" planning problem; 5) the causality-based representation and reasoner used in these studies allows us to compute optimal plans with concurrency, solve not only planning but also prediction/postdiction and diagnosis problems, use various search engines without modifying their algorithms. Thus, overall, our approach extends the classes of manipulation problems that can be solved and provides a

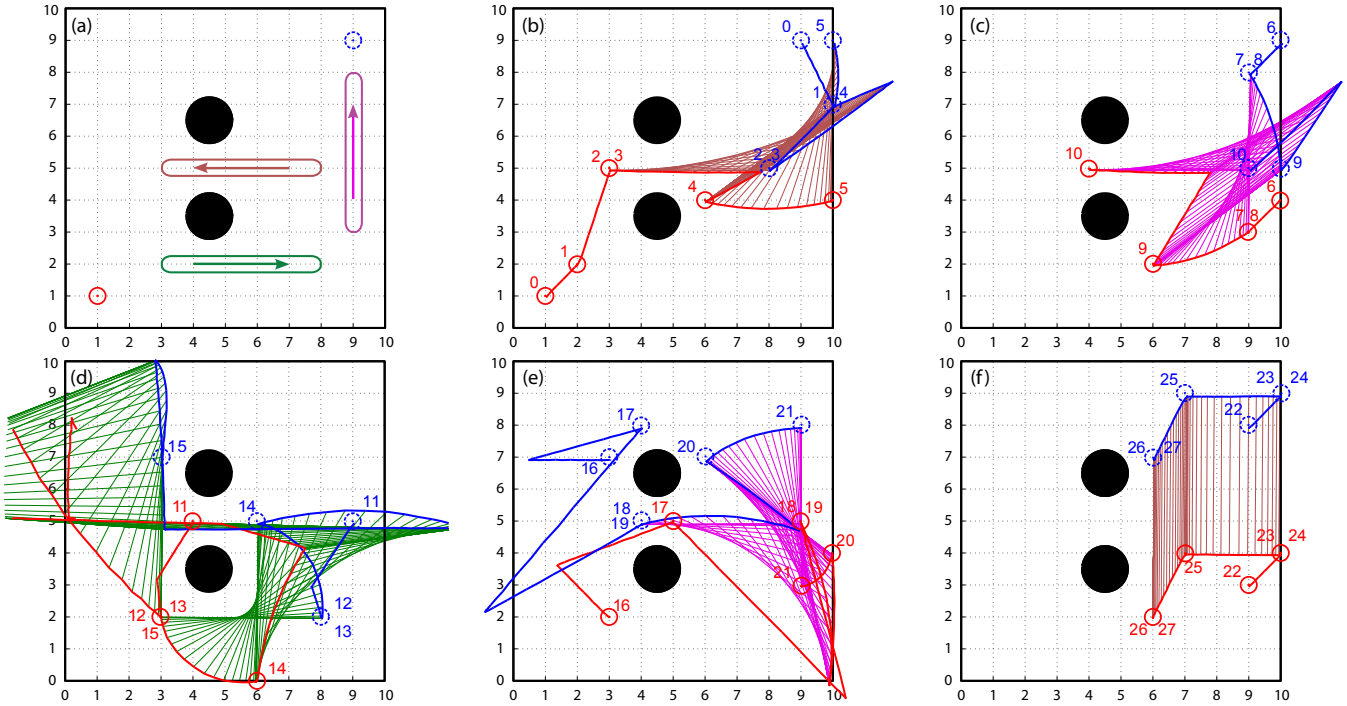

Fig. 2. (a) presents the initial state, while (b)–(f) illustrate the execution of Plan 2. Colors red and blue are associated with Robots 1 and 2. Circles indicate the positions of the robot end-effectors and circles' labels denote the time steps. Solid red and blue lines denote the trajectories of robot end-effectors. Brown, magenta and green lines denote Payloads 1–3. For instance, at Step 3, end-effectors of Robots 1 and 2 are located at `(3,5)` and `(8,5)` respectively, and robots hold Payload 1. At Step 4, end-effectors of Robots 1 and 2 are located at `(6,4)` and `(10,7)`, still holding Payload 1. Trajectory of Payload 1 moving from Step 3 to 4 is depicted in brown.

flexible and generic platform for integrating high-level action reasoning with low-level trajectory planning.

We have illustrated the usefulness of this approach with a physical implementation of a problem that involves two robots working for a common goal. In our experiments, we have observed that using parallel SAT solvers sometimes improve the computational efficiency by a factor of 100; these results motivate us to experiment with various search engines in the future. In particular, we consider using planners such as TLPlan that can take into account domain-specific information. Also during the experiments, we have observed that the calculated trajectories cannot always be robustly implemented due to uncertainties and noise that are unavoidable in a physical setup. Base on our preliminary work on execution monitoring [9], this motivates us to enrich our approach by integrating it into a monitoring framework to handle various kinds of execution failures (e.g., interventions and collisions with unknown objects). Updating/learning action domain descriptions from observations/failures, as in [4], is also a part of our future work.

References

[1] R. Alami, J.P. Laumond, and T. Simeon. Two manipulation planning algorithms. In *Proc. of Workshop on Algorithmic Foundations of Robotics*, pages 945–952, 1994.

[2] F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *AIJ*, 116(1–2):123–191, 2000.

[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] T. Eiter, E. Erdem, M. Fink, and J. Senko. Updating action domain descriptions. *AIJ*, 174(15):1172–1221, 2010.

[5] P. Eyerich, T. Keller, and B. Nebel. Combining action and motion planning via semantic attachments. In *Proc. of Workshop on Combining Action and Motion Planning at ICAPS*, 2010.

[6] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *JAIR*, 20:61–124, 2003.

[7] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. *AIJ*, 153:2004, 2004.

[8] F. Gravot, S. Cambon, and R. Alami. volume 15, chapter aSyMov:A Planner That Deals with Intricate Symbolic and Geometric Problems, pages 100–110. Springer, 2005.

[9] K. Haspalamutgil, C. Palaz, T. Uras, E. Erdem, and V. Patoglu. A tight integration of task planning and motion planning in an execution monitoring framework. In *Proc. of BTAMP*, 2010.

[10] K. Hauser and J.-C. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Proc. of BTAMP*, 2009.

[11] J. Hoffmann and B. Nebel. FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[12] L. P. Kaelbling and T. Lozano-Perez. Hierarchical planning in the now. In *Proc. of Workshop on Mobile Manipulation at ICRA*, 2010.

[13] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[14] Y. Koga and J. C.Latombe. On multi arm manipulation planning. In *Proc. of ICRA*, pages 945–952, 1994.

[15] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic, 1991.

[16] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[17] N. McCain and H. Turner. Causal theories of action and change. In *Proc. of AAAI/IAAI*, pages 460–465, 1997.

[18] E. Plaku and G. D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. of ICRA*, 2010.

[19] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, 2010.