Towards Integrating Hierarchical Goal Networks and Motion Planners to Support Planning for Human-Robot Teams

Vikas Shivashankar[1], Krishnanand N. Kaipa[2], Dana S. Nau[3], and Satyandra K. Gupta[4]

***Abstract*— Low-level motion planning techniques must be combined with high-level task planning formalisms in order to generate realistic plans that can be carried out by humans and robots. Previous attempts to integrate these two planning formalisms mostly used either Classical Planning or HTN Planning. Recently, we developed *Hierarchical Goal Networks* (HGNs), a new hierarchical planning formalism that combines the advantages of HTN and Classical planning, while mitigating some of the disadvantages of each individual formalism. In this paper, we describe our ongoing research on designing a planning formalism and algorithm that exploits the unique features of HGNs to better integrate task and motion planning. We also describe how the proposed planning framework can be instantiated to solve assembly planning problems involving human-robot teams.**

I. Introduction

Low-level motion planning techniques must be combined with high-level task planning formalisms in order to generate realistic plans that can be carried out by humans and robots. A representative application scenario is planning for fenceless assembly cells where robots can collaborate seamlessly with humans to perform assembly tasks. The advent of safer robots like Baxter and exteroceptive sensing based safety systems [1] are making this human robot collaboration (HRC) paradigm feasible. Some of the components required to achieve safe and efficient HRC include assembly sequence generation [2], task decomposition between human and robot [3], system state monitoring (tracking human, robot, and assembly parts) [4], automated instruction generation for human operations [5], ensuring human safety [1], and recovering from assembly errors [4]. In order to enable a coherent integration among these modules, a high-level planner, interleaved with low-level motion planners, is needed at several levels of the system hierarchy.

For example, given a CAD model of a product to be assembled, RRT based motion planning methods can be used to generate improved assembly precedence constraints [2]. In turn, these constraints can be compiled into a high-level planning problem. Humans and robots share complimentary strengths in performing assembly operations. Therefore, the planning framework must have the ability to incorporate this knowledge in order to decompose the tasks effectively. Further, an integral planner must be able to perform plan-repair in order to handle contingencies during assembly operations. The deviations causing these contingencies may be of two types: (1) low-level deviations in the geometric state without affecting the corresponding symbolic state (e.g., human places part in a wrong posture), which can be corrected at the motion planning level, or (2) deviations in the symbolic state itself (e.g., human picks an incorrect part, and an improved alternative assembly sequence may or may not exist), which needs to be corrected at both levels of planning.

Task planning formalisms typically used to achieve this integration are *Classical Planning* ([6], [7], [8], [9]) and *Hierarchical Task Network (HTN) Planning* ([10], [11], [12]). Classical planning is not scalable enough to handle complex problems since it can't leverage domain-specific knowledge, and HTNs impose stringent completeness requirements on domain models, which are difficult to guarantee in open, dynamic environments. Therefore, solutions attempting to integrate these AI planning techniques into domains involving real robots and humans are not very satisfactory.

In recent work [13], [14], we developed a new hierarchical planning formalism called *Hierarchical Goal Networks* (HGNs) that combines aspects of HTN and classical planning into a single framework. This is a hierarchical planning formalism similar to HTN planning, but operates over hierarchies of classical goals instead of tasks. It thus can combine the scalability and expressivity advantages of HTN planning with the heuristic-search/reasoning capabilities of Classical planning.

In this paper, we describe our ongoing work on integrating HGNs with motion planners to build better planning systems. The aims of this work are twofold:

1) Design a general-purpose planning-and-execution framework that combines HGN planning and execution-time plan-repair algorithms with off-the-shelf motion planners
2) Formulate this planning framework in the context of planning for human-robot teams in assembly cells.

The novelty in this approach comes from the interaction between the heuristic search techniques of the HGN and motion planners, and the particular way in which they guide each other towards high-quality solutions. The HGN planner, when invoking the motion planner, also passes to it an upper bound $\tau \texttt{cost}(a)$, where τ is a user-specified tolerance parameter. The motion planner, being a heuristic search planner, can detect when the lower bound on the best

[1]V. Shivashankar is a graduate student at Department of Computer Science, University of Maryland, College Park svikas@cs.umd.edu

[2]K. N. Kaipa is a Research Assistant Professor at Department of Mechanical Engineering, University of Maryland, College Park kkrishna@umd.edu

[3]D. S. Nau is a Professor at Department of Computer Science and Institute for Systems Research, University of Maryland, College Park nau@cs.umd.edu

[4]S. K. Gupta is a Professor at Department of Mechanical Engineering and Institute for Systems Research, University of Maryland, College Park skgupta@umd.edu

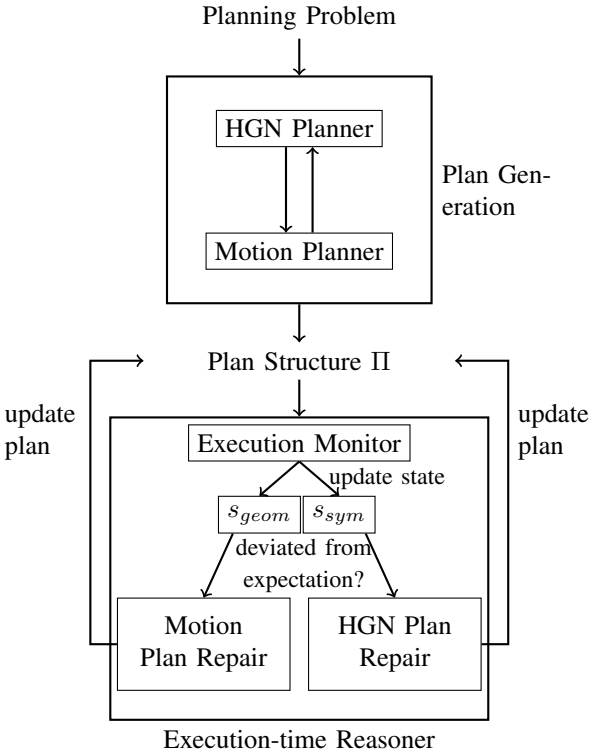

Fig. 1. System Architecture

possible solution it can generate exceeds this bound, and can return failure at that point. This is especially useful in cases when a bad goal configuration has been sampled, leading to unsolvable problems. Similarly, motion plan costs are also propagated to the task-planning level to update the heuristic estimates of the symbolic actions, and used to determine whether other actions might lead to better plan costs.

II. System Architecture

Figure 1 depicts our proposed architecture of the hybrid planning-and-execution system. This is similar to the conceptual model of planning and execution proposed by Nau [15], but specifically instantiated for planning domains of interest in this paper, with motion planning algorithms, HGN planning and plan-repair algorithms.

The system takes as input the planning problem P, which provides descriptions of the (1) initial state, (2) goals to be achieved, (3) base action models at the task-planning level, (4) control primitives at the motion planning level, and finally (5) procedures to translate between symbolic and geometric state descriptions. Section III describes this in more detail.

P is first input into the *Plan Generation* module. In this module, HGN planners and low-level motion planners interactively synthesize an executable plan structure Π that achieves the given goals when applied from the initial state of the system. See section IV for more details on this module.

The plan Π is then input into the *Execution-time Reasoner* module to (1) monitor plan execution, and (2) repair Π in case the deviations of the system state from expectation render the current plan inexecutable. These deviations can be of two types: (1) low-level deviations in the geometric state s_{geom} without affecting the corresponding symbolic state s_{sym} (e.g., human places part in a wrong posture) which can be corrected at the motion planning level, or (2) deviations in the symbolic state itself (e.g., human picks an incorrect part, and an improved alternative assembly sequence may or may not exist) which needs to be corrected at both levels of planning. In both cases, Π is then updated with the new plan and execution is resumed. These considerations are discussed in Section V.

III. Planning Formalism

Formalisms for HGN planning, motion planning, and integration between the two planning layers are presented below:

A. Hierarchical Goal Network Planning

Task Planning Domain. We define the task planning model M_{TP} as a five-tuple $(V_D, V_C, \mathcal{O}, \mathcal{M}, \gamma)$. V_D is the set of discrete state variables in the domain; they evaluate to either `true`/`false` (e.g. `empty(arm1)=true`) or to a discrete object in the domain (e.g. `pos(robot1)=loc1`). V_C on the other hand represents the set of continuous state variables in the domain, which can evaluate to a real number (e.g. `cpos(robot1,joint3)=4.78`).We refer to assignments to variables in V_D and V_C as symbolic and geometric states; a system state s is a pair (s_{sym}, s_{geom}).

$\mathcal{O}$ represents the set of primitive operator models in the domain, which are model actions that are executable in a single step at the task planning level. Each $o \in \mathcal{O}$ is a four-tuple $(\texttt{name}(o), \texttt{pre}(o), \texttt{eff}(o), \texttt{cost}(o))$, where:

- $\texttt{name}(o)$ represents the action name and its parameters $\texttt{arg-list}(o) = (V_o^D, V_o^C)$, which represent the discrete and continuous arguments to o respectively;
- $\texttt{pre}(o) = (\texttt{pre}_o^D, \texttt{pre}_o^C)$ represents the preconditions of o, representing conditions over the variables in V_o^D and V_o^C respectively;
- $\texttt{eff}(o)$ represents effects over V_o^D that o produces as a result of its application;
- $\texttt{cost}(o)$ is a function that takes as input $\texttt{arg-list}(o)$ and returns a non-negative cost.

$\mathcal{M}$ represents the set of HGN methods, which models domain-specific knowledge that suggests ways to decompose goals into subgoals. Each $m \in \mathcal{M}$ is a four-tuple $(\texttt{name}(m), \texttt{post}(m), \texttt{pre}(m), \texttt{subgoals}(m))$, where:

- $\texttt{name}(m)$ represents the method name and its parameters $\texttt{arg-list}(m) = (V_m^D, V_m^C)$, which represent the discrete and continuous arguments to m respectively;
- $\texttt{post}(m)$ represents the goal condition defined over V_m^D that m is relevant for; m is thus triggered only when $\texttt{post}(m)$ is part of the goal condition being solved for;
- $\texttt{pre}(m) = (\texttt{pre}_m^D, \texttt{pre}_m^C)$ represents the conditions under which m is applicable, represented over the variables in V_m^D and V_m^C respectively;
- $\texttt{subgoals}(m)$ is a sequence of subgoals defined over V_m^D that needs to be achieved enroute to achieving $\texttt{post}(m)$.

Finally, γ represents the state transition function. A ground instance a of an operator o is applicable in a state s if s satisfies $\texttt{pre}(a)$; the resulting state $s' = \gamma(s, a)$ reassigns the state variables according to the assignments in $\texttt{eff}(a)$.

Task Planning Problems. A *goal network* is a way to represent the objective of satisfying a partially ordered sequence of goals (hence one can think of it as a particular kind of temporally extended goal). Formally, it is a pair $gn = (T, \prec)$ such that:

- T is a finite nonempty set of nodes;
- each node $t \in T$ contains a *goal* g_t that is a DNF (disjunctive normal form) formula over discrete state variable assignments;
- $\prec$ is a partial order over T.

A *HGN planning problem* is a triple $P = (M_{TP}, s_0, gn)$, where M_{TP} is a task planning model, s_0 is the initial state, and $gn = (T, \prec)$ is a goal network.

Definition 1. *The set of* solutions *for P is defined as follows:*

Case 1. *If T is empty, the empty plan is a solution for P.*

Case 2. *Let t be a node in T that has no predecessors. If $s_0 \models g_t$, then any solution for $P' = (M_{TP}, s_0, (T', \prec'))$ is also a solution for P, where $T' = T - \{t\}$, and $\prec'$ is the restriction of $\prec$ to T'.*

Case 3. *If action a is applicable in s_0 and π is a solution for $P'' = (M_{TP}, \gamma(s_0, a), gn)$, then $a \circ \pi$ is a solution for P.*

B. Motion Planning

Let $\chi = \mathbb{R}^d$, $d \leq |V_C|$ be the configuration space of the system. Let χ_{obs} be the obstacle region; thus $\chi_{free} = \chi \setminus \chi_{obs}$ represents the obstacle-free space. A motion planning problem is a triple $P = (\chi_{free}, x_0, \chi_{goal})$ where x_0 is an element of χ_{free} and the goal χ_{goal} is a subset of χ_{free}.

A path $\sigma : [0, 1] \rightarrow \mathbb{R}^d$ is a valid solution to P if (1) it is *continuous*, (2) it is *collision-free*, i.e. $\sigma(\tau) \in \chi_{free}$ for all $\tau \in [0, 1]$, and (3) the boundary conditions are satisfied, i.e. $\sigma(0) = x_0$ and $\sigma(1) \in \chi_{goal}$.

C. Connecting Task and Motion Planning Levels

As shown in Section III-A, HGN planning, being a task planning formalism, is primarily concerned with search for a path through the symbolic state space induced by the discrete state variables V_D, while motion planners search for paths through the geometric state space induced by the continuous variables in V_C. In order to connect these two levels of planning, we must first provide a way to switch between these two different state spaces: i.e. we must provide a way to (1) generate candidate geometric states consistent with a symbolic state, and conversely (2) generate the symbolic state corresponding to a given geometric state.

We assume the provision of the following domain-specific procedures as part of the domain description:

- Gen_{sym}which takes as input a geometric state s_{geom} and generates the corresponding symbolic state s_{sym}
- Gen_{geom}which takes as input a symbolic state s_{sym} and generates a candidate geometric state s_{geom}.

There is a lot of flexibility in the definition of Gen_{sym} and Gen_{geom}. For instance, Gen_{sym} could be a classifier that is learnt during the training stage [9] or a rule-based generator. Similarly, Gen_{geom} could be domain-specific *geometric suggesters* [10], [11] or a simple sampling-based generator [12].

Thus, the overall planning problem P is a 3-tuple $(\langle M_{TP}, \chi_{free}, \mathsf{Gen}_{sym}, \mathsf{Gen}_{geom}\rangle, s_0, gn)$ where s_0 and gn are the initial state and the goal network respectively.

Definition 2. *We can now provide the overall definition of solutions for a planning problem P as follows. Let π_{sym} be a solution of the underlying HGN planning problem $P_{sym} = (M_{TP}, s_0, gn)$.*

Case 1. *If π_{sym} is empty, then $\Pi = \langle\rangle$ is a solution for P.*

Case 2. *Let $\pi_{sym} = a \circ \pi'_{sym}$. Furthermore, let s_1^{sym} be the symbolic state after a is executed. Let s_0^{geom} be the projection of s_0 onto the variables in V_C, and s_1^{geom} be the geometric state generated by Gen_{geom} for s_1. If there exists a valid solution σ to the motion planning problem $(\chi_{free}, s_0^{geom}, s_1^{geom})$ and Π' is a solution to $P' = (\langle M_{TP}, \mathsf{Gen}_{sym}, \mathsf{Gen}_{geom}\rangle, \langle s_1^{sym}, s_1^{geom}\rangle, gn)$, then $\Pi = \sigma \circ \Pi'$ is a solution to P.*

IV. The GoDeL Algorithm

Algorithm 1 is the GOAL DECOMPOSITION WITH LANDMARKS[1] (GoDeL) planning algorithm, a HGN planning algorithm [14], modified to integrate motion planning into the plan generation.

A. Task Planning in GoDeL

Lines 3 – 6 specify the base cases of GoDeL. If these are not satisfied, the algorithm nondeterministically chooses a goal g with no predecessors and generates $\mathcal{U}$, all method and operator instances applicable in s and relevant to g. It then chooses a $u \in \mathcal{U}$ with the lowest heuristic value to progress the search (Lines 7 – 10). If u is an action, the state is progressed to $\gamma(s, u)$ (Line 11) and u is further refined into an executable motion plan (Lines 13–25). Section IV-B talks about this in more detail. Else, the subgoals of u are added to G, adding edges to preserve the total order imposed on $\texttt{subgoals}(u)$ (Line 23). In either case, GoDeL is invoked recursively on the new planning problem (Lines 21 and 25).

B. Motion Planning in GoDeL

Motion planners are used to refine actions at the task-planning level into executable motion plans. GoDeL can use any off-the-shelf motion planner; the only requirement is that the motion planner should be able to provide an *anytime lower bound* during the search. In other words, the motion planner at any time should be able to provide a lower bound on the best solution its going to find henceforth. Therefore heuristic motion planners such as ARA* [16], R* [17], which do (weighted) A* search are ideally suited for use here.

[1]Simplified for ease of presentation; this version doesn't include the auxiliary planning techniques needed to work with incomplete/incorrect methods, as these aspects are orthogonal to the focus of this paper. Refer to [14] for the full version of GoDeL.

Algorithm 1: A nondeterministic version of GoDeL. Initially, (D, s, G) is the planning problem, and π is $\langle\rangle$.

```
1  Procedure GoDeL(D, s, G, π)
2  begin
3      if G is empty then return π
4      nondeterministically choose a goal formula g in G
       without any predecessors
5      if s ⊨ g then
6          return GoDeL(D, s, G − {g}, π)
7      U ← {(u, 0) : u is an operator/method instance
       applicable to s and relevant to g}
8      sort U in ascending order of h_HGN(s, u, G)
9      while U is not empty do
10         remove (u, n) with min h value (= h_u) from U
11         if u is an action then
12             for i=n to NumSamples do
13                 σ_{s,u} ← RefineAction(D, s, u)
14                 if σ_{s,u} ≠ failure then
15                     h_new = |σ_{s,u}| − cost(u) + h_u
                       insert (⟨u, σ_{s,u}⟩, −1) into U with
                       h = h_new
16                     if i < NumSamples then insert
                       (u, i) into U, updating its heuristic
                       value using h_new
17                     break
18
19
20         else if (u, σ_{s,u}) is action, mot. plan pair then
21             res1 ← GoDeL (D, UpdateState(s′, σ_{s,u}),
               G, π ∘ ⟨u, σ_{s,s′}⟩)
22             if res1 ≠ failure then return res1
23         else
24             res1 ← GoDeL(D, s, subgoals(u) ∘ G, π)
25         if res1 ≠ failure then return res1
26     return failure
27 end
```

Algorithm 2: Procedure to refine an action at the task-planning level into an executable motion plan. It takes as input the action u and the current state s, and outputs either a valid motion plan $\sigma_{s,u}$ or failure. It also takes as input two user-specified parameters, τ and NumSamples.

```
1  Procedure RefineAction(D, s, u)
2  begin
3      s′ ← γ(s, u)
4      x_{s′} ← Gen_geom(s, s′_sym)
5      σ_{s,u} ← MotionPlanner(χ_free, s_geom,
       x_{s′}, τ × cost(u))
6      return σ_{s,u}
7  end
```

The motion planning in GoDeL is parametrized by two extra user-specified constants, τ and NumSamples. The model of any action u at the task-planning level has a cost function which provides a rough estimate of how much any refinement of u would cost. τ is an upper bound on the number of times by which the motion plan cost is allowed to overshoot $\texttt{cost}(u)$. NumSamples is the maximum number of geometric configurations the motion planner is invoked on for a given symbolic state before backtracking.

When it reaches an action u, GoDeL invokes RefineAction to generate a motion plan that refines u (Line 13). RefineAction picks a geometric configuration consistent with u's effects, and passes it to MotionPlanner along with the upper bound of $\tau \times \texttt{cost}(u)$. MotionPlanner thus either returns a plan within this bound or returns failure. This feature in particular helps in cases when no motion plan exists, which RefineAction can now automatically detect and stop the motion planner.

When GoDeL receives the motion plan $\sigma_{s,u}$, it inserts it into $\mathcal{U}$ with heuristic value of $\sigma_{s,u}$ (Line 15). It also reinserts the original action u annotated with the number of samples remaining with the heuristic value updated with $\sigma_{s,u}$ (Line 16), as this is a more accurate estimate of the cost of u.

The purpose of this reinsertion is the following: suppose $\texttt{cost}(u)$ was overly optimistic and the real estimate of u, $|\sigma_{s,u}|$, is significantly larger, we would like to give other actions a chance to refine to cheaper motion plans. If $\sigma_{s,u}$ is indeed the best option, it would be removed from $\mathcal{U}$ in the next iteration, and GoDeL will be recursively called in the state resulting from it (Line 21). In this way, we are tightly integrating task and motion planning search spaces by using results from one to help direct the search in the other.

This process is repeated until either each action has been refined NumSamples times and failed, in which case the planner backtracks, or a recursive call on the state resulting from a particular motion plan generates a valid plan, in which case GoDeL succeeds and returns the generated plan.

V. Execution-time Reasoning Techniques

Once the *Plan Generation* module has generated a candidate plan Π, this is then provided to the *Execution-time Reasoner* module for plan execution. This module consists of three components:

- **Execution Monitor**: this keeps track of the current value of the geometric state s_{geom} of the system via sensors. In turn, the corresponding symbolic state s_{sym} is also continuously updated via the Gen_{sym} procedure and monitored for deviations.
- **Motion Plan Repair Algorithm**: If the Execution Monitor reports a deviation from s_{geom} without a corresponding deviation for s_{sym}, incremental plan-repair algorithms at the motion planning level such as LPA* [18], D* [19], etc can be used to repair the motion plan and restore executability.
- **HGN Plan Repair Algorithm**: If either a deviation from s_{sym} was detected or the deviation from s_{geom} could not be resolved at the motion planning level,

incremental plan-repair algorithms at the task-planning level can be used to restore executability.

We are currently working on an incremental HGN plan-repair algorithm that can repair symbolic plans during execution in the face of exogenous events and even changing goals. Our approach involves repurposing HGN heuristics used for planning to guide the plan repair process by identifying promising repair points in the plan structure. This differentiates it from other hierarchical plan-repair algorithms proposed in the literature [20], [21] which do blind plan repair, which can lead to severely suboptimal solutions.

VI. Related Work

Most of the related work on integration of task and motion planning techniques either use *Classical Planning* or *Hierarchical Planning* techniques.

Classical Planning. The seminal work in integrating Classical Planning and Motion Planning is by Cambon et al [6]. They use the FF planner as a heuristic estimator to guide the search of the motion planner. Erdem et al [7] integrate task and motion planning by extending the $\mathcal{C}+$ action description language to allow incorporation of external predicates which allow interfacing with external programs; thus one could run arbitrary domain-specific geometric reasoners and motion planners as part of precondition checks of primitive actions. Similarly, Dornhege et al [8] propose a similar framework to specify 'semantic attachments' as part of PDDL domain descriptions and extend the FF planner to generate plans for such problems. Burbridge et al [9] suggest a novel machine learning approach to interface between the discrete and continuous state spaces, and use this to integrate motion planners with classical planners. Our approach differs from these by the use of hierarchical planning techniques which allow increased expressivity and scalability, and by the particular ways in which flow of heuristic estimates from the task planner to the motion planner and vice-versa allow for more robust search and explicit plan cost optimization.

Hierarchical Planning. Wolfe et al [12] combine HTN planners with motion planners by discretizing the choice of geometric states via finite sampling during refinement of a symbolic action. Another line of work [10], [11] builds a special-purpose regression-based hierarchical planner that they combine with motion planners. Our approach differs from these techniques by the use of HGN planners, which allow for use of heuristics during hierarchical decomposition, and as explained in the previous paragraph, by the novel way in which heuristic estimates from each level of planning guide the other.

VII. Planning for Assembly Cells

One of our aims is to ground our proposed architecture (described in Fig. 1) in a manufacturing domain. Figure 2 shows a snapshot of such a system. The shop floor is divided into four regions: (1) Part storage, (2) Tool storage, (3) Subassemblies building, and (4) Final assembly.

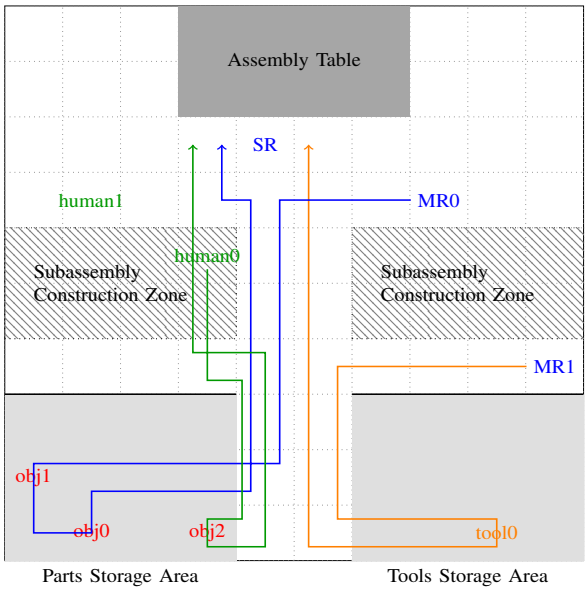

Fig. 2. Sample assembly planning problem:`MR0` and `MR1` are mobile robots, `human0` and `human1` are humans, and `SR` is a static robot. The goal is to assemble `obj0`, `obj1` and `obj2` together using `tool0`. One of the solutions induced by the task planning model is also shown.

The proposed system takes a CAD model of the product that needs to be assembled as input and performs the following:

- **Assembly Precedence Constraints Generation**: automatically generate assembly precedence constraints and detect useful subassemblies using techniques in [2] and compile this into an HGN planning problem.
- **Plan Generation**: Use the integrated HGN and motion planning algorithm to generate an executable plan structure Π – actions to be performed by humans can be provided at the symbolic level, while those performed by robots need to be refined into motion plans. Factors that determine whether a particular task needs to be executed by a human or robot can be modeled in the action and method preconditions; the planner, based on the particulars of a given task, can use these conditions to generate compatible assignments. Higher-level protocols that need to be followed (such as reserving a tool before use) can be modeled as HGN methods.
- **Execution-time Reasoning**: Execute Π, while continuously monitoring the geometric state of the system using sensors, both vision-based and otherwise, that monitor locations of workers, tools and parts, as well as other information such as battery levels of robots, stress in robot arms while carrying heavy loads, etc. The system will repair Π using HGN and motion plan repair algorithms, when deviations from both the expected symbolic and geometric states are observed.

We model the action models of the domain in **P**lanning **D**omain **D**escription **L**anguage (PDDL), a standard language to encode task-level planning domain descriptions.

The base actions in this domain are as follows:

1) `Worker-Move (workerId start-loc goal-loc)`
2) `Worker-Pickup-Object (workerId objId obj-loc)`
3) `Worker-Putdown-Object (workerId objId goal-loc)`
4) `Worker-Move-Near-Object (workerId current-loc objId)`
5) `Worker-Move-Near-Location (workerId current-loc goal-loc)`
6) `Mark-Worker-Free (workerId)`
7) `Mark-Worker-Busy (workerId)`
8) `Asmbl-Objs (workerId obj1Id obj2Id)`

These action models, in addition to modeling causal preconditions and effects, also model procedural and geometric preconditions over the continuous variables; for instance, the `Worker-Pickup-Object` action checks if the object is graspable by the worker, its weight is within the carrying capacity of the worker, etc. Thus, the planner can flexibly assign humans or robots to execute actions based on their compatibility with the preconditions. If a robot is assigned, only then is the action is further refined using motion planners. If a human is assigned, the system then passes the symbolic action directly for execution; however, since the system has no control over the exact geometric state that results from the action execution, the planner chooses one nondeterministically and relies on the plan-repair system to repair the plan if needed.

VIII. Conclusion and Future Work

The primary aim of this work is to build a general-purpose planning-and-execution system that enable long-term autonomous behavior by human-robot teams. One of the components needed to achieve this is a planning system that tightly integrates task and motion planning. The main contribution of this paper lies in leveraging the advantages of HGN planning, which allows combining hierarchical decomposition with heuristic search, to achieve this tight integration. The combined planning algorithm allows for using search information from the task planner to guide the search for a high-quality motion plan by the motion planner, as well as using search results from the motion planner to revisit decisions made by the task planner. Currently, we are working on implementing the planning algorithm and empirically evaluating the efficacy of this approach on assembly planning problems. As future work:

- **Plan Repair**. We are currently working on developing an HGN execution semantics and an execution-time plan-repair algorithm; we are interested in integrating this with plan-repair techniques at the motion planning level, thus enabling a combined plan-repair system.
- **Anytime Plan Improvement**. Since we have heuristic search capabilities at both planning levels, an interesting extension would be in quickly generating a plan upfront, and then improving these plans in an anytime fashion; this would allow for exploiting the full power of anytime heuristic motion planners [16].

Acknowledgements. This work was supported in part by ARO grant W911NF1210471 and ONR grants N000141210430 and N000141310597. The information in this paper does not necessarily reflect the position or policy of the funders.

References

[1] C. Morato, K. N. Kaipa, B. Zhao, and S. K. Gupta, "Toward safe human robot collaboration by using multiple kinects based real-time human tracking," *ASME Journal of Computing and Information Science in Engineering*, vol. 14, no. 1, p. 011006, 2014.

[2] C. Morato, K. N. Kaipa, and S. K. Gupta, "Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters," *Computer-Aided Design*, vol. 45, no. 11, pp. 1349 – 1364, 2013.

[3] K. N. Kaipa, C. Morato, J. Liu, and S. K. Gupta., "Human-robot collaboration for bin-picking tasks to support low-volume assemblies," in *Human-Robot Collaboration for Industrial Manufacturing Workshop, held at Robotics: Science and Systems Conference (RSS 2014)*, 2014.

[4] C. Morato, K. N. Kaipa, J. Liu, and S. K. Gupta., "A framework for hybrid cells that support safe and efficient human-robot collaboration in assembly operations," in *ASME Computers and Information Engineering Conference*, 2014.

[5] K. N. Kaipa, C. Morato, B. Zhao, and S. K. Gupta., "Instruction generation for assembly operations performed by humans," in *ASME Computers and Information Engineering Conference*, 2012.

[6] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *I. J. Robotic Res.*, vol. 28, no. 1, pp. 104–126, 2009.

[7] 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 *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 4575–4581.

[8] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *ICAPS*, 2009.

[9] C. Burbridge and R. Dearden, "An approach to efficient planning for robotic manipulation tasks," in *ICAPS*, 2013.

[10] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *ICRA*, 2011, pp. 1470–1477.

[11] D. Hadfield-Menell, L. P. Kaelbling, and T. Lozano-Perez, "Optimization in the now: Dynamic peephole optimization for hierarchical planning," in *ICRA*, 2013.

[12] J. Wolfe, B. Marthi, and S. J. Russell, "Combined task and motion planning for mobile manipulation," in *ICAPS*, 2010, pp. 254–258.

[13] V. Shivashankar, U. Kuter, D. S. Nau, and R. Alford, "A hierarchical goal-based formalism and algorithm for single-agent planning," in *AAMAS*, 2012, pp. 981–988.

[14] V. Shivashankar, R. Alford, U. Kuter, and D. S. Nau, "The godel planning system: A more perfect union of domain-independent planning and hierarchical planning," in *IJCAI*, 2013.

[15] D. S. Nau, "Current trends in automated planning," *AI Magazine*, vol. 28, no. 4, pp. 43–58, 2007.

[16] M. Likhachev, D. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artif. Intell.*, 2008.

[17] M. Likhachev and A. Stentz, "R* search," in *AAAI*, 2008, pp. 344–350.

[18] M. Likhachev and S. Koenig, "A generalized framework for lifelong planning A* search," in *ICAPS*, 2005, pp. 99–108.

[19] M. Likhachev, D. Ferguson , G. Gordon, A. T. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," 2005.

[20] N. F. Ayan, U. Kuter, F. Yaman, and R. Goldman, "Hotride: Hierarchical ordered task replanning in dynamic environments," in *Proceedings of the ICAPS Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution.*, 2007.

[21] J. Bidot, B. Schattenberg, and S. Biundo, "Plan repair in hybrid planning," in *KI 2008: Advances in Artificial Intelligence*, 2008.