Levels of Integration between Low-Level Reasoning and Task Planning*

Peter Schüller, Volkan Patoglu, Esra Erdem

Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey

Abstract— We provide a systematic analysis of levels of integration between discrete high-level reasoning and continuous low-level reasoning to address hybrid planning problems in robotics. We identify four distinct strategies for such an integration: (i) low-level checks are done for all possible cases in advance and then this information is used during plan generation, (ii) low-level checks are done exactly when they are needed during the search for a plan, (iii) first all plans are computed and then infeasible ones are filtered, and (iv) by means of replanning, after finding a plan, low-level checks identify whether it is infeasible or not; if it is infeasible, a new plan is computed considering the results of previous low-level checks. We perform experiments on hybrid planning problems in robotic manipulation and legged locomotion domains considering these four methods of integration, as well as some of their combinations. We analyze the usefulness of levels of integration in these domains, both from the point of view of computational efficiency (in time and space) and from the point of view of plan quality relative to its feasibility. We discuss advantages and disadvantages of each strategy in the light of experimental results and provide some guidelines on choosing proper strategies for a given domain.

I. INTRODUCTION

Successful deployment of robotic assistants in our society requires these systems to deal with high complexity and wide variability of their surroundings to perform typical everyday tasks robustly and without sacrificing safety. Consequently, there exists a pressing need to furnish these robotic systems not only with discrete high-level reasoning (e.g., task planning, diagnostic reasoning) and continuous low-level reasoning (e.g., trajectory planning, deadline and stability enforcement) capabilities, but also their tight integration resulting in hybrid planning.

Motivated by the importance of hybrid planning, recently there have been some studies on integrating discrete task planning and continuous motion planning. These studies can be grouped into two, where integration is done at the search level or at the representation level. For instance, [1], [2], [3], [4], [5], [6] 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. Each one of these approaches presents a specialized combination of task and motion planning at the search level, and does not consider a general interface between task and motion planning.

On the other hand, [7], [8], [9], [10] integrate task and motion planning by considering a general interface between them, using "external predicates/functions", which are predicates/functions that are computed by an external mechanism, e.g., by a C++ program. The idea is to use external predicates/functions in the representation of actions, e.g., for checking the feasibility of a primitive action by a motion planner. So, 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 means of external predicates/functions. [7], [9] apply this approach in the action description language $\mathcal{C}+$ [11] using the causal reasoner CCALC [12]; [10] applies it in Answer Set Programming (ASP) [13], [14] using the ASP solver CLASP [15]; [8] extends the planning domain description language PDDL [16] to support external predicates/functions (called semantic attachments) and modifies the planner FF [17] accordingly.

In these approaches, integration of task and motion planning is achieved at various levels. For instance, [9], [10] do not delegate all sorts of feasibility checks to external predicates as in [7], [8], but implements only some of the feasibility checks (e.g., checking collisions of robots with each other and with other objects, but not collisions of objects with each other) as external predicates and use these external predicates in action descriptions to guide task planning. For a tighter integration, feasibility of task plans is checked by a dynamic simulator; in case of infeasible plans, the planning problem is modified with respect to the causes of infeasibilities, and the task planner is asked to find another plan.

In this paper, our goal is to better understand how much of integration between high-level reasoning and continuous low-level reasoning is useful, and for what sort of robotic applications. For that, we consider integration at the representation level, since this approach allows a modular integration via an interface, external predicates/functions, which provides some flexibility of embedding continuous low-level reasoning into high-level reasoning at various levels. Such a flexible framework allowing a modular integration is important for a systematic analysis of levels of integration.

We identify four distinct strategies to integrate a set of continuous feasibility checks into high-level reasoning, grouped into two: *directly integrating* low-level checks into high-level reasoning while a feasible plan is being generated, and generating candidate plans and then *post-checking* the feasibility of these candidate solutions with respect to the low-level checks. For direct integration we investigate two

methods of integration: (i) low-level checks are done for all possible cases in advance and then this information is used during plan generation, (ii) low-level checks are done when they are needed during the search for a plan. For post-checking we look at two methods of integration: (iii) all plans are computed and then infeasible ones are filtered, (iv) by means of replanning, after finding a plan, low-level checks identify whether it is infeasible or not; if it is infeasible, a new plan is computed considering the results of previous low-level checks. We consider these four methods of integration, as well as some of their combinations; for instance, some geometric reasoning can be integrated within search as needed, whereas some temporal reasoning is utilized only after a plan is computed in a replanning loop. Considering each method and some of their combinations provide us different levels of integration.

To investigate the usefulness of these levels of integration at representation level, we consider 1) the expressive formalism of HEX programs for describing actions and the efficient HEX solver dlvhex to compute plans, and 2) the expressive formalism of ASP programs for describing actions and the efficient ASP solver CLASP to compute plans. Unlike the formalisms and solvers used in other approaches [7], [8], [9], [10], that study integration at representation level, HEX [18] and dlvhex [19] allow external predicates/functions to take relations (e.g., a fluent describing locations of all objects) as input without having to explicitly enumerate the objects in the domain. Other formalisms and solvers allow external predicates/functions to take a limited number of objects and/or object variables as input only, and thus they do not allow embedding all continuous feasibility checks in the action descriptions. In that sense, the use of HEX programs with dlvhex, along with the ASP programs with CLASP enriches the extent of our experiments.

We perform experiments on planning problems in a robotic manipulation domain (like in [9]) and in a legged locomotion domain (like in [20], [21]). The robotic manipulation domain involve 3D collision checks and inverse kinematics, whereas legged locomotion examples involve stability checks and reachability checks. We analyze the usefulness of levels of integration in these domains, both from the point of view of computational efficiency (in time and space) and from the point of view of plan quality relative to its feasibility.

II. LEVELS OF INTEGRATION

Assume that we have a task planning problem instance H (consisting of an initial state S_0, goal conditions, and action descriptions) in a robotics domain, represented in some logic-based formalism. A history of a plan $\langle A_0, \ldots, A_{n-1} \rangle$ from the given initial state S_0 to a goal state S_n computed for H consists of a sequence of transitions between states: $\langle S_0, A_0, S_1, A_1, \ldots, S_{n-1}, A_{n-1}, S_n \rangle$. A *low-level continuous reasoning module* gets as input, a part of a plan history computed for H and returns whether this part of the plan history is feasible or not with respect to some geometric, dynamic or temporal reasoning.

For example, if the position of a robot at step t is represented as $robot\_at(x, y, t)$ and the robot's action of moving to another location (x', y') at step t is represented as $move\_to(x', y', t)$, then a motion planner could be used to verify feasibility of the movement $\langle robot\_at(x, y, t), move\_to(x', y', t), robot\_at(x', y', t+1) \rangle$. If duration of this action is represented as well, e.g., as $move\_to(x, y, duration, t)$, then the low-level module can find an estimate of the duration of this movement relative to the trajectory computed by a motion planner, and it can determine the feasibility of the movement $\langle robot\_at(x, y, t), move\_to(x', y', t), robot\_at(x', y', t+1) \rangle$ by comparing this estimate with $duration$.

Let L denote a low-level reasoning module that can be used for the feasibility checks of plans for a planning problem instance H. We consider four different methods of utilizing L for computing feasible plans for H, grouped into two: *directly integrating* reasoning L into H, and *post-checking* candidate solutions of H using L.

For *directly integrating* low-level reasoning into plan generation, we propose the following two levels of integration:

- **PRE** – *Precomputation* We perform all possible feasibility checks of L that can be required by H, in advance. For each failed check, we identify the actions that cause the failure, and then add a constraint to the action descriptions in H ensuring that these actions do not occur in a plan computed for H. We then try to find a plan for the augmented planning problem instance H^{pre}. Clearly, every plan obtained with this method satisfies all low-level checks.

- **INT** – *Interleaved Computation* We do not precompute anything, but we interleave low-level checks with high-level reasoning in the search of a plan: for each action considered during the search, the necessary low-level checks are immediately performed to find out whether including this action will lead to an infeasible plan. An action is included in the plan only if it is feasible. The results of feasibility checks of actions can be stored not to consider infeasible actions repeatedly in the search of a plan. Plans generated by interleaved computation satisfy all low-level checks.

Let us denote by L_{PRE} and L_{INT} the low-level checks directly integrated into plan generation, with respect to PRE and INT, respectively.

Alternatively, we can integrate low-level checks L with H, by means of *post-checking* candidate solutions of H relative to L. We propose the following two methods to perform post-checks on solution candidates:

- **FILT** – *Filtering*: We generate all plan candidates for H. For each low-level check in L, we check feasibility of each plan candidate and discard all infeasible candidates.

- **REPL** – *Replanning*: We generate a plan candidate for H. For each low-level check in L, we check feasibility of the plan candidate. Whenever a low-level check fails, we identify the actions that cause the failure, and then
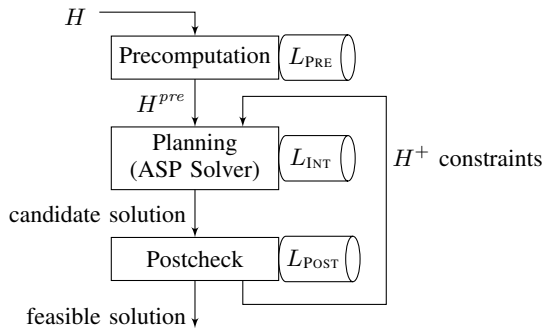
Fig. 1. Components and data flow.

add a constraint to H ensuring that these actions do not occur in a plan computed for H. We generate a plan candidate for the updated planning problem instance H^+ and do the feasibility checks. We continue with generation of plan candidates and low-level checks until we find a feasible plan, or find out that such a feasible plan does not exist.

Let us denote by L_{POST} the low-level checks done after plan generation, with respect to FILT or REPL.

Figure 1 shows the hybrid planning framework we use in this paper to compare different levels of integration, and combinations thereof, on robotics planning scenarios. In particular, Fig. 1 depicts computational components: Precomputation extends the problem instance H using a low-level reasoning module L_{PRE}, Planning integrates a low-level reasoning module L_{INT} into its search for a plan candidate for the problem instance H^{pre} generated by Precomputation. Postcheck uses a low-level module L_{POST} to verify solution candidates (using FILT or REPL) and to potentially add constraints H^+ to the input of Planning.

In our systematic analysis of levels of integration, we do consider this hybrid framework by disabling some of its components. For instance, to analyze the usefulness of PRE, we disable the other integrations (i.e., $L_{\text{INT}} = L_{\text{POST}} = \emptyset$); to analyze the usefulness of a combination of PRE and FILT, we disable other integrations (i.e., $L_{\text{INT}} = \emptyset$).

III. METHODOLOGY

We investigate the usefulness of levels of integration as described above, considering two orthogonal properties: solution quality and planning efficiency. We quantify these properties as follows.

A. Solution Quality

If some low-level module L is not integrated into the planning process, some plan candidates will be infeasible due to failed low-level checks of L. We quantify solution quality by measuring the number of feasible and infeasible plan candidates generated by the search for a plan. This way we obtain a measure that shows how *relevant* a given low-level check is for plan feasibility. Note that with the FILT approach an infeasible plan candidate simply causes a new plan to be generated, while with REPL an infeasible plan

candidate causes computation of additional constraints, and a restart of the plan search.

Tightly connected to the number of feasible and infeasible solution candidates is the number of low-level checks that is performed until finding the first feasible plan, and until finding all feasible plans.

B. Planning Efficiency

We quantify planning efficiency by measuring the time required to obtain the first feasible plan, and the time to enumerate all feasible plans. (Note that this includes proving that no further plan exists.)

Independent from the number of low-level checks, the duration of these external computations can dominate the overall planning cost, or it can be negligible. Therefore we measure not only the number of computations of low-level modules but also the time spent in these computations.

IV. DOMAINS AND EXPERIMENTAL SETUP

For our empirical evaluation we use the Robotic Manipulation and the Legged Locomotion domains. Both require hybrid planning. We next give an overview of the domains, their characteristics, and scenarios we used.

A. Robotic Manipulation

We consider a cooperative robotic manipulation problem, as in [9], where two robots arrange elongated objects in a space that contains obstacles. The manipulated objects can only be carried cooperatively by both robots, objects must not collide with each other or the environment, similarly robots must not collide with each other.

A large part of collision checks between objects can already be realized in the high-level representation, however certain checks require usage of geometric models. Collision-freeness between robots for particular collaborative actions can only be determined using low-level geometric reasoning and is not represented in the world model.

Therefore we use two low-level reasoning components to check collision-freeness: given end effector positions of both arms, the L_{rob} module checks whether this combination of positions is collision-free, similarly L_{pay} checks whether an object can be located at a particular pair of coordinates without a collision with the environment. (Additional motion planning to check feasibility of movement from one position to another one can be added as a third check but was omitted in our experiments.) We experiment with 10 instances (over a 11×11 grid) that require plans of upto 20 (average 9.2) steps, and involving up to 58 (average 25.1) actions.

B. Legged Locomotion

In the Legged Locomotion domain, a robot with high degrees of freedom must find a plan for placing its legs and moving its center of mass (CM) in order to move from one location to another one.

For the purpose of studying integration of geometric reasoning with high-level task planning, we created a planning formulation for a four-legged robot that moves on a 10×10

grid. Some grid locations are occupied and must not be used by the robot. Starting from a given initial configuration, the robot must reach a specified goal location where all legs are in contact with the ground.

As legged robots have high degrees of freedom, legged locomotion planning deals with planning in a high-dimensional space. We use a planning problem that is of similar complexity as has been investigated in climbing [21] and walking [22] robots. We also require a feasibility check of leg placement actions. We allow concurrent actions, i.e., moving the center of mass while detaching a leg from the ground, if this does not cause the robot to lose its balance.

We use a low-level reasoning component that determines whether the robot is in a balanced stable equilibrium (L_{bal}), given its leg positions and the position of its CM. We realize this check by computing the support polygon of legs that are currently connected to the ground, and by checking if CM is within that polygon. For these checks we use the `boost::geometry` library to compute a convex hull of all leg positions, and then check whether CM is located within that convex hull.

A second low-level module determines if leg positions are realistic wrt. the position of CM, i.e., if every leg can reach the position where it is supposed to touch to the ground. This check (L_{leg}) is realized as a distance computation between coordinates of legs and CM.

C. Domain Characteristics and Notable Differences

The domains we experiment with exhibit various differences in their characteristics, and such a variety allows us to get practically more relevant results. The most important differences between these two domains are as follows.

Complexity of low-level reasoning. In Legged Locomotion we use a C++ geometric library to perform basic geometric operations which are sufficient for computing check results.

In Robotic Manipulation, object collision checking L_{pay} operates on 3D models of objects and environments additionally L_{rob} requires inverse kinematics to determine the joint configuration of each robot reaching a certain point before performing collision checks between arms.

Hence, in Legged Locomotion, each low-level check requires less time and memory than in Robotic Manipulation. Note that in both domains, we transform discrete 2D grid points to continuous 3D coordinates for low-level reasoning.

Information relevant for low-level reasoning. In Legged Locomotion, we consider problem instances over a 10×10 grid. L_{leg} is a check over two coordinates, therefore there are 10^4 possible L_{leg} checks. The balance check L_{bal} is a totally different situation: we have an input of four leg coordinates and one CM coordinate, therefore, there are 10^{10} possible L_{bal} checks. Such a large number of checks makes precomputation infeasible.

In Robotic Manipulation, both low-level checks are over coordinate pairs on a 11×11 grid; therefore, there are $11^4 \cdot 2 = 29282$ low-level checks.

TABLE I

EFFICIENCY COMPARISON

Integration Method	Overall Time		Low-Level Reasoning	
	FIRST sec	ALL sec	time ALL sec	count ALL #
Robotic Manipulation (10 instances)				
FILT	1716 [2]	1877 [2]	39	724
REPL	2007 [2]	3242 [3]	7	139
PRE	888	974	238	29282
INT	1007	1086	0	467
Legged Locomotion (averages over 20 instances)				
FILT	2434 [5]	3091 [8]	1139	35888
REPL	1345	4192 [9]	12	458
INT	80	133	21	171109
L_{leg}: L_{bal}:				
PRE FILT	2395 [6]	3046 [8]	1272	39354
PRE REPL	1160	4142 [9]	9	324
PRE INT	65	107	23	50677

Numbers in square brackets count timeouts for FIRST resp. ALL.

Based on the number of low-level checks, precomputation for Legged Locomotion seems feasible for only one of the two low-level modules (L_{leg}), while for Robotic Manipulation we can apply precomputation for both low-level computations. Indeed, precomputation for Legged Locomotion can be done in less than 1 second, and for Robotic Manipulation in 238 seconds.[1]

V. EXPERIMENTAL RESULTS

We applied different integration methods to 20 Legged Locomotion and 10 Robotic Manipulation instances of varying size and difficulty.

Tables I and II present results for

- **FIRST**: obtaining the first feasible plan, and for
- **ALL**: obtaining all (maximum 10000) feasible plans.

In our experiments, we use a timeout of 2 hours (7200 seconds) after which we stop computation and take measurements until that moment.

We also limit the number of enumerated plans to 10000 plans. The measurements for enumerating up to 10000 plans reveal information about solution quality and provides a more complete picture of the behavior of each method: one method might find a feasible solutions very fast by chance, whereas finding many or all solutions fast by chance is unlikely.

A. Time Measurements

Table I shows measurements regarding planning efficiency and effort spent in low-level reasoning.

Firstly, it is clear that PRE and INT— the direct integration methods — outperform FILT and REPL— the post-checking methods: for Robotic Manipulation, only PRE and INT are able to enumerate all solutions within the given time limit; for Legged Locomotion, only INT and the PRE/INT combination enumerates all solutions.

Comparing the times required by PRE and INT, we see that PRE is more efficient for Robotic Manipulation (888 sec vs 1007 sec on average), which is mainly due to an efficient precomputation technique (see Section V-E).

Even though PRE performs better than INT, it spends more time in low-level reasoning, hence high-level reasoning is faster there; we can explain this by a more constrained search space (low-level check results constraint the search).

After PRE and INT, the next best choice is REPL: it finds solutions to 8 out of 10 instances in Robotic Manipulation, and it finds solutions to all instances for Legged Locomotion, whereas FILT has the same number of timeouts in the Manipulation domain and 5 timeouts for Legged Locomotion. In addition to that, we can see that REPL spends little time in low-level checks compared to other approaches. This is because REPL performs many restarts of the high-level planner which causes it to spend a disproportionate amount of time in high-level planning. Nevertheless, REPL shows its robustness by finding solutions to all but 2 instances.

Finally, FILT fails to find solutions for 7 instances in total which clearly makes it the worst-performing method. The time results for Robotic Manipulation suggest that FILT may be a bit faster than REPL; this may be an effect of some easy instances in that domain where replanning spends more time by reinitialization, than FILT spends by iterating over many similar infeasible solutions. Therefore, even in that domain, we would not suggest to use FILT, as it might — by chance, as low-level reasoning cannot give feedback to high-level reasoning — fail to find a feasible solution for a long time.

B. Effort of Low-Level Reasoning

In Robotic Manipulation, while attempting to enumerate all solutions, FILT performs only 724 low-level checks compared to 29282 checks of PRE. Similarly, in Legged Locomotion, FILT performs 35888 checks and fails to enumerate all solutions for 8 of 20 instances, while INT enumerates all solutions while performing more (171109) low-level checks. Note that these numbers (the last column of Table I) indicate *distinct* low-level reasoning tasks as we cache low-level check results. These numbers show that FILT encounters a small fraction of the low-level checks that are needed to verify all solutions in INT. Caching in fact allows FILT to verify much more actions than INT (numbers not shown), however the number of *distinct* checks (numbers shown) is higher in INT. We conclude that INT traverses the solution space much more efficiently.

In Legged Locomotion, low-level checks depend on a large part of the candidate plan, so caching is not as effective as in Robotic Manipulation. This, together with the fact that in FILT the high-level is not guided by low-level checks, causes the FILT approach to spend more time in low-level reasoning than other approaches.

Note that, to obtain a reasonable comparison between PRE and the other approaches, we include times and counts of precomputed low-level checks in Table I (which explains the large values for low-level computations in these rows).

TABLE II
SOLUTION QUALITY COMPARISON

Integration Method	Infeasible Candidates		Plans found	Feasible Plans
	FIRST #	ALL #	ALL #	ALL %
Robotic Manipulation (averages over 10 instances)				
FILT	586	11787	622	<0.1
REPL	11	38	621	94.2
PRE	0	0	652	100.0
INT	0	0	652	100.0
Legged Locomotion (averages over 20 instances)				
FILT	11282	35487	360	1.0
REPL	28	68	250	78.6
INT	0	0	1394	100.0
L_{leg}: L_{bal}:				
PRE FILT	10938	39116	340	0.9
PRE REPL	31	69	255	78.7
PRE INT	0	0	1394	100.0

C. Solution Quality

Methods PRE and INT do not generate infeasible solution candidates, as they use all low-level checks already in search.

If we compare the number of infeasible solution candidates of FILT and REPL in Robotic Manipulation, we observe that FILT generates mainly infeasible solution candidates compared to the number of feasible solutions (11787 vs 622) while REPL creates only 38 infeasible candidates while enumerating 621 feasible plans.

In Legged Locomotion, the results for FILT are similar, however REPL performs a bit worse than in Robotic Manipulation with 250 infeasible candidates compared to 68 feasible solutions. A possible reason for this difference could be the same reason why PRE is not feasible in that domain: there is a large amount of possible inputs to L_{bal} compared to the other low-level checks we used. Due to the large input space, each failed L_{bal} check constrains the search space only by a small amount, so REPL produces more infeasible solutions than in Robotic Manipulation.

D. Memory Usage

We measured peak memory usage over the whole runtime of each instance. Interleaved computation with the dlvhex solver (columns with INT) requires an average of around 2000MB, the maximum stays below 4000MB. For non-interleaved computations, GRINGO and CLASP were connected with low-level checks using Python scripts. These approaches require around 400MB of memory with a maximum below 1000MB.

E. Combination of PRE with other methods

As shown in the Legged Locomotion experiments, PRE can be combined with other approaches. In our experiments we observe that adding PRE increases efficiency.

However, PRE adds a fixed cost to solving because it needs to precompute many points. Depending on efficiency of low-level computations, even if there are few possible

input combinations to low-level modules, precomputation might be infeasible.

It can be feasible to precompute a large set of related low-level checks more efficiently than performing a check for each possible input combination. In our experiments we implemented such a *dedicated precomputation* method: for Robotic Manipulation it requires 238 seconds in total, while calling individual checks requires 1361 seconds in total. Hence without dedicated efficient precomputation, PRE performs much worse than INT.

VI. DISCUSSION AND CONCLUSION

Our experiments suggest the following conclusions. If robust and highly complex reasoning is required, and if this reasoning is done frequently (so that performance gains will become relevant) then using full interleaved reasoning (INT) is the only good option. INT has the best performance with respect to run times, and it can enumerate most solutions compared to other approaches. The reason is that INT uses only those low-level checks which are necessary (they are computed on demand) and therefore does not overload the solver with redundant information (as PRE does). Furthermore, INT considers failed checks in the search process and thereby never picks an action where it is known that the action will violate a low-level check. This is similar as in the REPL approach, but much more efficient as the integration is much tighter compared to REPL. However, the performance of INT comes at a price: (a) it requires more memory than generating solution candidates and checking them afterwards, and (b) it requires a solver that allows for interacting with the search process in a tight way, usually through an API that has to be used in a sophisticated way if it shall be efficient.

If reasoning operates on a manageable amount of inputs, such that precomputation is a feasible option, then PRE is a good choice. In our Robotic Manipulation experiments, PRE outperforms all other methods, which is partially due to our using a dedicated efficient precomputation tool (see Section V-E). In Legged Locomotion, combining PRE with other methods also increased efficiency.

The FILT approach performs the worst, because nothing guides the search into the direction of a feasible solution; FILT is not robust and enumerates many infeasible solutions.

If both PRE and INT are not possible[2] then REPL should be used; this approach does not have the same performance as INT and PRE, however it is a very robust approach as it is guided by its wrong choices — we can think of the constraints that are added for failed low-level checks as the approach 'learning from its mistakes'. The benchmark results for Legged Locomotion clearly show the robustness of REPL compared to FILT: the former finds solutions for all problems, the latter only for 15 out of 20 instances.

A possible improvement to REPL could be to let it enumerate a certain amount of solutions to gather more constraints, then add all these constraints and restart the search. This is

a hybrid approach between FILT and REPL. Selecting the right moment to abort enumeration and restart the solver is crucial to the performance of such a hybrid approach, and we consider this a worthwhile subject for future investigations.

REFERENCES

[1] F. Gravot, S. Cambon, and R. Alami, *Robotics Research The Eleventh International Symposium*, ser. Springer Tracts in Advanced Robotics. Springer, 2005, vol. 15, ch. aSyMov:A Planner That Deals with Intricate Symbolic and Geometric Problems, pp. 100–110.

[2] K. Hauser and J.-C. Latombe, "Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries," in *Workshop on Bridging the Gap between Task and Motion Planning at ICAPS*, 2009.

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

[4] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *Proc. of ICRA*, 2010, pp. 5002–5008.

[5] J. Wolfe, B. Marthi, and S. Russell, "Combined task and motion planning for mobile manipulation," in *International Conference on Automated Planning and Scheduling*, 2010.

[6] E. Plaku, "Planning in discrete and continuous spaces: From LTL tasks to robot motions," in *Proc. of TAROS*, 2012, pp. 331–342.

[7] 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.

[8] A. Hertle, C. Dornhege, T. Keller, and B. Nebel, "Planning with semantic attachments: An object-oriented view," in *Proc. of ECAI*, 2012, pp. 402–407.

[9] 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 *International Conference on Robotics and Automation*, 2011, pp. 4575–4581.

[10] E. Erdem, E. Aker, and V. Patoglu, "Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution," *Intelligent Service Robotics*, vol. 5, pp. 275–291, 2012.

[11] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, "Nonmonotonic causal theories," *AIJ*, vol. 153, p. 2004, 2004.

[12] N. McCain and H. Turner, "Causal theories of action and change," in *Proc. of AAAI/IAAI*, 1997, pp. 460–465.

[13] V. Lifschitz, "What is answer set programming?" in *Proc. of AAAI*. MIT Press, 2008, pp. 1594–1597.

[14] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming at a glance," *Commun. ACM*, vol. 54, no. 12, pp. 92–103, 2011.

[15] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "clasp: A conflict-driven answer set solver," in *Proc. of LPNMR*, 2007, pp. 260–265.

[16] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[17] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, 2001.

[18] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits, "A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 90–96.

[19] T. Eiter, G.Ianni, R.Schindlauer, and H.Tompits, "Effective integration of declarative rules with external evaluations for Semantic-Web reasoning," in *Proc. of ESWC*, 2006.

[20] T. Bretl, S. Lall, J.-C. Latombe, and S. M. Rock, "Multi-step motion planning for free-climbing robots," in *Algorithmic Foundations of Robotics VI*, ser. Springer Tracts in Advanced Robotics, vol. 17. Springer, 2005, pp. 59–74.

[21] T. Bretl, S. M. Rock, J.-C. Latombe, B. Kennedy, and H. Aghazarian, "Free-climbing with a multi-use robot," in *ISER*, ser. Springer Tracts in Advanced Robotics, M. H. A. Jr. and O. Khatib, Eds., vol. 21. Springer, 2004, pp. 449–458.

[22] K. K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *I. J. Robotic Res*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

---

[2]This can be the case if the state of inputs is large, and additionally we need to use a certain planner software that does not support interleaved computation.