A Systematic Analysis of Levels of Integration between Low-Level Reasoning and Task Planning

Peter Schüller, Volkan Patoglu, Esra Erdem

***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 the plan 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 housekeeping and legged locomotion domains considering these four methods of integration, as well as some of their combinations. We analyze the usefulness of different levels of integration in these domains, both from the point of view of computational efficiency 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, [12, 13, 17, 22, 23, 21] 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, [4, 15, 7, 8] 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. [4, 7] apply this approach in the action description language $\mathcal{C}+$ [11] using the causal reasoner CCALC [20]; [8] applies it in Answer Set Programming (ASP) [19, 3] using the ASP solver CLASP [10]; [15] extends the planning domain description language PDDL [9] to support external predicates/functions (called semantic attachments) and modifies the planner FF [16] accordingly.

In these approaches, integration of task and motion planning is achieved at various levels. For instance, [7, 8] do not delegate all sorts of feasibility checks to external predicates as in [4, 15], 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

This work is partially supported by TUBITAK Grant 111E116. Peter Schüller is supported by TUBITAK 2216 Research Fellowship.

P. Schüller, V. Patoglu, E. Erdem are with Faculty of Engineering and Natural Sciences, Sabancı University, İstanbul, Turkey {peterschueller,esraerdem,vpatoglu}@sabanciuniv.edu

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 [4, 15, 7, 8], that study integration at representation level, HEX [5] and dlvhex [6] 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 housekeeping domain (like in [8]) and in a legged locomotion domain (like in [2, 1]). The housekeeping examples involve geometric reasoning (i.e., collision checks), temporal reasoning (i.e., restrictions on the total duration of plans), 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 add

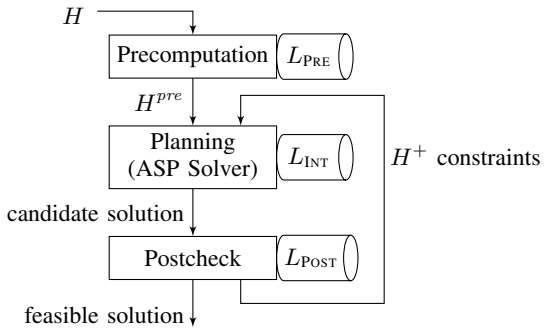

Fig. 1. Components and data flow.

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, considering solution quality and computation times.

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 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 Housekeeping and the Legged Locomotion domains. Both require hybrid planning. We next give an overview of the domains, their characteristics, and scenarios we used.

A. Housekeeping

In Housekeeping, multiple autonomous robots collaboratively tidy up a house, by putting items in rooms to their proper places, for example dirty dishes are put into the dishwasher, books into the bookcase, and pillows into the bed. The domain we use is similar to [8]: robots can move from certain locations to other locations, they can attach to and detach from objects, some objects must be carried collaboratively as they are heavy. Detecting that the plan is the shortest possible plan, and creating a plan that has a feasible trajectory for each robot, requires geometric and temporal low-level reasoning.

We use two external low-level reasoning components: checking whether a path exists from one place in the world to another place ($L_{\exists}$) and computing the quasi-optimal path to estimate the time that is required for moving along that path (L_{opt}). Both checks are realized using the RRT (Rapidly Exploring Random Trees) approach [18] in C++; in $L_{\exists}$ we immediately abort sampling when we find a solution, for L_{opt} we sample for a longer amount of time and apply path smoothening to obtain shorter paths. If the motion planner does not return a solution in a given amount of time (which is determined over experiments), the action is considered infeasible; in such cases feasible high-level plans (if there exists one) can be missed.

We perform tests on 17 Housekeeping instances (over 8×8 grid) with up to 5 robots and 8 objects, which require plans up to 12 steps (average 7) and up to 41 actions in a single plan (average 22).

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.

TABLE I
EFFICIENCY COMPARISON

Integration Method	Overall Time FIRST sec	Overall Time ALL sec	Low-Level Reasoning time ALL sec	Low-Level Reasoning count ALL #
Housekeeping (17 instances)				
FILT	5972 (14)	6684 (14)	13	26
REPL	3126 (2)	5641 (13)	307	657
PRE	3479	6272 (12)	3256	8192
INT	1046	4488 (10)	534	1168
Legged Locomotion (averages over 15 instances)				
FILT	4663 (9)	4919 (10)	672	25834
REPL	2876 (5)	5073 (10)	5	207
INT	84	170	8	12344
L_{leg}: L_{bal}:				
PRE FILT	4193 (6)	4943 (9)	1732	62489
PRE REPL	3081 (5)	5191 (10)	349	10105
PRE INT	408	432	351	20538

Numbers in brackets count timeouts for FIRST resp. ALL.

TABLE II
SOLUTION QUALITY COMPARISON

Integration Method	Infeasible Candidates FIRST #	Infeasible Candidates ALL #	Plans found ALL #	Feasible Plans ALL %
Housekeeping (averages over 17 instances)				
FILT	2538	107476	28	<0.1
REPL	44	107	1575	93.6
PRE	0	0	43790	100.0
INT	0	0	348770	100.0
Legged Locomotion (averages over 15 instances)				
FILT	5253	25698	8	<0.1
REPL	20	118	15	11.3
INT	0	0	56	100.0
L_{leg}: L_{bal}:				
PRE FILT	24740	52482	12	<0.1
PRE REPL	16	94	17	15.3
PRE INT	0	0	56	100.0

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 [1] and walking [14] 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.

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

These two domains exhibit various differences in their characteristics.

Complexity of low-level reasoning. In Housekeeping, the low-level reasoning modules are realized in an executable that performs motion planning to check feasibility of actions. On the contrary, in Legged Locomotion we use a C++ geometric library to perform basic geometric operations which are sufficient for computing check results.

Information relevant for low-level reasoning. In Housekeeping, both low-level modules operate on a pair of coordinates for each robot movement action, hence the amount of information required for a low-level check is limited. We use instances with a 8×8 grid, therefore precomputation of $L_{\exists}$ and L_{opt} is feasible; each check must be done for $8^4 = 4096$ possible inputs. In Legged Locomotion we use a 10×10 grid. L_{leg} is a check over two coordinates as in the Housekeeping domain, therefore there are 10^4 possible inputs, precomputation is feasible. However, for the balance check L_{bal} we have an input of four leg coordinates and one CM coordinate, therefore in our grid there are 10^{10} possible inputs which makes precomputation infeasible. Hence for Legged Locomotion we apply precomputation only to L_{leg}.

Independence of low-level modules. In Housekeeping, a successful check of path existence $L_{\exists}$ is a prerequisite for the more costly shortest path check L_{opt}. In Legged Locomotion, L_{leg} and L_{bal} check independent geometric concerns. In Housekeeping experiments we therefore apply the same integration method to $L_{\exists}$ and L_{opt} and thus consider four different settings. Different from that, in Legged Locomotion we consider six settings and vary the level of integration along two dimensions, configuring L_{leg} independently from L_{bal}.

V. EXPERIMENTAL RESULTS

We applied each of the above scenarios to 17 Housekeeping and 15 Legged Locomotion instances of varying size and difficulty. As some instances have many solutions, and some methods proved to be very slow compared to others, we use a timeout of 2 hours (7200 seconds) after which we stop computation and take measurements until that moment. All experiments are performed on a Linux server with 32 2.4GHz Intel® E5-2665 CPU cores and 64GB memory. We used the ASP solver dlvhex for INT experiments, and CLASP with GRINGO for the rest of the experiments.

Tables I and II present results for

- **FIRST**: obtaining the first feasible plan,
- **ALL**: obtaining all feasible plans or hitting the timeout.

In most practical applications, performance of FIRST will be more relevant. However ALL reveals additional information about solution quality, and it provides a more robust picture of the behavior of each method: one method might find a first solutions very fast by chance, whereas finding many or all solutions fast by chance is an unlikely event.

The left column shows which method of integration was used.

A. Time Measurements

For obtaining the first solution, we present the average over all runs where not finding a solution was counted as the timeout of 7200 seconds.

The INT approach clearly outperforms other approaches for finding the first solution. FILT is worst in terms of runtime and in terms of solutions obtained. When comparing replanning and precomputation, we see that REPL takes less time but does not find solutions for some instances, while PRE takes more time, always finds some solution, and enumerates much more solutions than PRE. Therefore, the larger effort of PRE in low-level reasoning might pay off in terms of solved instances.

B. Solution Quality

Table II shows that 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 Housekeeping, we observe that FILT generates too many infeasible candidates compared to the number of solutions (107476 vs. 28) while REPL creates a moderate amount of infeasible solution candidates compared to the number of feasible plans (107 vs. 1575).

In Legged Locomotion the results for FILT are similar, however REPL performs worse than in Housekeeping (it produces 118 infeasible candidates and 15 solutions). We can explain this difference in solution quality by the larger amount of possible inputs to L_{bal} compared to the other low-level checks: each failed L_{bal} check can constrain the search space for candidate solutions only by a small amount, so REPL cannot avoid as many infeasible solutions in Legged Locomotion as in Housekeeping.

To ensure that timeouts do not affect our analysis of solution quality, we also performed an analysis over those instances without timeouts; these results are not shown in the tables above due to space restrictions. For this set of instances, we can find all 28 solutions for Housekeeping, and all 8 solutions for Legged Locomotion. FILT is the worst option in terms of solution quality: it generates 2539 infeasible candidates for Housekeeping and 3948 for Legged Locomotion (FILT/FILT). REPL is better: 14 in feasible candidates for Housekeeping and 48 for Legged Locomotion (REPL/REPL); note that REPL still performs better for Housekeeping examples.

C. Effort Spent in Low-Level Checks

In Legged Locomotion, the FILT approach spends most of its time in low-level checks, as it is not guided by earlier failed checks. Moreover these checks depend on a large part of the candidate plan, so caching is not effective. Contrarily, REPL is guided by failed checks and spends more time searching for solution candidates and less time in low-level checks.

Using FILT with a precomputed L_{leg} makes it much faster: we observe that PRE/FILT finds solutions to 3 more instances than FILT/FILT, performs much more low-level checks than FILT/FILT and spends more time in these checks. The reason for that is the more constrained search space given precomputed leg range checks, this makes the solver find more solution candidates in the same time, therefore more time can be spent for low-level reasoning.

In Housekeeping the situation looks very different: FILT requires only very little effort in low-level checks. This might seem unintuitive at first, however the reason is that both $L_{\exists}$ and L_{opt} depend on a small set of inputs, therefore the basic cache we implement can mark many solution candidates as infeasible, and the effort spent in low-level checks stays low. As FILT is not guided by earlier failed checks, the solver generates many similar solutions; the cache has the effect that these similar and infeasible solutions can be discarded without considerable effort in external computations.

To obtain a fair comparison between precomputation and the other approaches, we include times and counts of precomputed low-level checks in Table I (which explains their large values).

VI. DISCUSSION AND CONCLUSION

Our experiments suggest the following. 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 runtimes, 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. Therefore INT will not be usable for certain applications and solvers.

If reasoning operates on a manageable amount of inputs, such that precomputation is a feasible option, then PRE is a good choice. In the Housekeeping experiments, even when spending half the time until the timeout in precomputation, PRE outperforms FILT and REPL when it comes to finding a solution and enumerating solutions. It does not outperform INT, however, and only simple reasoning tasks with few possible input parameters can be handled with PRE.

As we did in the Legged Locomotion experiments, PRE can be combined with other approaches. If we reduce the times in Table I by the time required for doing the precomputation (346 seconds), then we see that adding PRE to other approaches always makes these approaches faster. For instance, INT/INT requires 84 seconds to find the first solution, while PRE/INT requires 408 seconds, where 346 seconds are precomputation time such that the actual search only takes 63 seconds. Therefore, we conclude that precomputation can be considered a possibility (if it is applicable).

The filtering approach turns out to be the worst, because nothing guides the search into the direction of a feasible solution. Therefore, FILT enumerates many infeasible solutions and might find no feasible solution for a very long time.

If precomputation and full interleaving is not possible (either because it is too complicated to setup in practice, or if no suitable reasoner for INT is available) then REPL should be used. It does not have the same performance as INT and PRE have, however it is a very robust approach, as it is guided by its wrong choices — and we can think of the constraints that are added for failed low-level checks as the approach 'learning from its mistakes'. The benchmark results clearly show the robustness of REPL compared to FILT: it has only 2 timeouts for finding a feasible plan for the 17 Housekeeping instances, while FILT has 14 timeouts.

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 worthwile subject for future investigations.

REFERENCES

[1] Timothy Bretl, Stephen M. Rock, Jean-Claude Latombe, Brett Kennedy, and Hrand Aghazarian. Free-climbing with a multi-use robot. In *Proc. of ISER*, 2004.

[2] Timothy Bretl, Sanjay Lall, Jean-Claude Latombe, and Stephen M. Rock. Multi-step motion planning for free-climbing robots. In *Proc. of WAFR*, pages 59–74, 2005.

[3] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[4] Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem, and Volkan Patoglu. Bridging the gap between high-level reasoning and low-level control. In *Proc. of LPNMR*, 2009.

[5] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In *Proc. of IJCAI*, 2005.

[6] Thomas 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.

[7] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. of ICRA*, 2011.

[8] Esra Erdem, Erdi Aker, and Volkan Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics*, 5:275–291, 2012.

[9] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003.

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

[11] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *AIJ*, 153:2004, 2004.

[12] Fabien Gravot, Stephane Cambon, and Rachid Alami. In *Proc. of ISRR*, pages 100–110, 2005.

[13] Kris Hauser and Jean-Claude Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Proc. of BTAMP*, 2009.

[14] Kris Hauser, Timothy Bretl, Jean-Claude Latombe, Kensuke Harada, and Brian Wilcox. Motion planning for legged robots on varied terrain. *I. J. Robotic Res*, 27 (11-12):1325–1349, 2008.

[15] Andreas Hertle, Christian Dornhege, Thomas Keller, and Bernhard Nebel. Planning with semantic attachments: An object-oriented view. In *Proc. of ECAI*, 2012.

[16] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *J. Artif. Intell. Res.*, 14:253–302, 2001.

[17] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. of ICRA*, pages 1470–1477, 2011.

[18] James J. Kuffner Jr and Steve M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. of ICRA*, pages 995–1001, 2000.

[19] Vladimir Lifschitz. What is answer set programming? In *Proc. of AAAI*, pages 1594–1597, 2008.

[20] Norman McCain and Hudson Turner. Causal theories of action and change. In *Proc. of AAAI/IAAI*, 1997.

[21] Erion Plaku. Planning in discrete and continuous spaces: From ltl tasks to robot motions. In *Proc. of TAROS*, pages 331–342, 2012.

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

[23] Jason Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. In *Proc. of ICAPS*, 2010.