Integrating Symbolic and Geometric Planning for Mobile Manipulation

Christian Dornhege and Marc Gissler and Matthias Teschner and Bernhard Nebel
Department of Computer Science
Georges-Köhler-Allee 52
Freiburg, Germany
{dornhege,gisslerm,teschner,nebel}@informatik.uni-freiburg.de

Abstract — Mobile manipulation requires to solve multiple subproblems. One is planning in high-dimensional configuration spaces, that we approach in this work. We decompose the manipulation problem into a symbolic and a geometric part. The symbolic part is implemented as a classical symbolic planner that tightly integrates a geometric planner enabling us to efficiently generate correct plans. A probabilistic roadmap planner constitutes the geometric part. During the computation of the roadmap we utilize proximity queries to determine non-colliding configurations and to verify collision-free paths between configurations accurately and efficiently. We demonstrate experiments in two scenarios, one of these being the manipulator dexterity test scenario that was used in NIST's response robot evaluation in Disaster City.

Keywords: *mobile manipulation, symbolic planning, manipulation planning*

I. Introduction

Mobile manipulation requires to solve multiple subproblems: accurate three dimensional world perception in unknown environments, geometric planning in high-dimensional configuration spaces and - in the case of tele-operation - human-robot-interfacing. In this paper, we are concerned with the planning problem and therewith also provide a helpful solution for operator assistance. Manipulation problems arise in autonomous robot operation as well as for tele-operated robots, where often the manipulator has to operate in confined scenarios and usually the tele-operator's comprehension of a scene is restricted by the camera perspective. Moreover, a manipulator's kinematics are not trivial to control. The best case in applied robotic systems is "tool center point control" that enables an operator or algorithm to control the manipulator's tool in cartesian coordinates. Alternate solutions are master-slave controllers or direct control of individual joints. All of those methods cannot easily prevent unintended collisions with the environment, especially in the usual camera-in-hand setting.

One such difficult scenario is the manipulator dexterity test proposed by NIST [1] and used in the response robot evaluation in Disaster City [2] (see figure 1). Our experiments show, among other problems, successful collision free planning in a reconstruction of this scenario.

We approach the manipulation problem by decomposing it into a symbolic and a geometric part. The symbolic view uses solution steps as *pick-up(box)* or *put-down(box, table)*. Thus, it presents a natural representation that can easily be solved by classical symbolic planners. The geometric view consists of the full problem description representing the manipulator's kinematics and a three dimensional scene description that is solved using a trajectory planner that computes collision free trajectories for the symbolic counterparts.

Fig. 1. The manipulator dexterity test. Left: Original setup as used in Disaster City. Note the horizontal bars that can be used to mount vertical boards extremely restricting the robot's workspace. Right: The reconstruction we use in our experiments.

The usual approach to decomposition is to hierarchically combine symbolic and geometric planners in a top-down or bottom-up manner. Following the first strategy, a plan is generated first that is then executed assuming the symbolic abstraction was correct. Following the second strategy, all geometric information is precomputed and then provided to the symbolic planner. Obviously, both strategies are not ideal. Therefore, we integrate the planners tightly, as we proposed in our previous work [7]. Thus, the low-level geometric planner can provide information to the high-level symbolic planner *during* the planning process. However, it is only evoked when it seems relevant to the high-level planner. Contrary to the hierarchical decomposition and combination, a particular choice on the symbolic level can lead the low-level planner to detecting failure and requesting to backtrack immediately.

To integrate information about special-purpose reasoning into symbolic planning we use *semantic attachments* in a planning domain description. Predicate symbols of the domain description used in grasp and put-down actions have such a semantic attachment, which means that these are not uninterpreted predicate symbols but that the truth values for atomic ground formulas are specified by an *external mechanism*: the

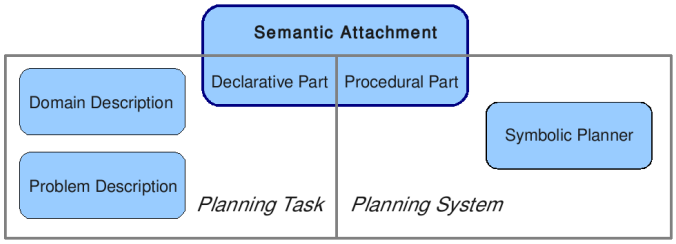

Fig. 2. Extending planning tasks by modules to planning tasks with semantic attachments.

trajectory planner.

We decompose the manipulation planning problem by using the common solution of viewing a manipulation path as a combination of transit and transfer paths [4]. Transit paths model *pick-up* actions that move the manipulator towards a possible grasp position, resulting in the object being grasped. Transfer paths move an object from one place to another and correspond to *put-down* actions on the symbolic level resulting in the manipulator releasing the object. To evaluate the applicability of these actions by semantic attachments, we run a probabilistic roadmap planner (PRM) [5]. During the computation of the roadmap we utilize proximity queries to determine non-colliding configurations and to verify collision-free paths between configurations accurately and efficiently similar to the approach by Schwarzer et al. [6]. Additionally we can give distance bounds to account for inaccurate world modeling and imprecise execution leading to safer plans.

II. Symbolic planning with semantic attachments

A symbolic planner decides the applicability of actions by evaluating conditions over state variables. Semantic attachments are external procedural reasoning modules (in the following just called *modules*) that compute the valuations of state variables at planner run-time. The symbolic planner itself is mostly unaffected by this extension. Under the hood of the module, though, complex computations can be performed that transcend the capabilities of the planner.

In order to integrate semantic attachments into a planner we propose the architecture shown in Figure 2. Semantic attachments consist of a *declarative part* that describes their use in the planning domain, i.e., their symbolic use in preconditions and effects of planning operators. Additionally, they have a *procedural part* which is the actual algorithm for computing the value of a state variable and which is directly included into and called by the planner as a shared library and which themselves may access the planning state through callback functions provided by the symbolic planner.

We use two kinds of semantic attachments that can be part of operators: *Condition checker* modules that can test whether some complex operator precondition is satisfied, and *effect applicator* modules which may compute changes to (several) numeric state variables.

To actually use semantic attachments in classical planning, it is necessary to extend the description language for planning tasks. In our previous work, we extended the planning domain description language (PDDL) to support semantic attachments leading to PDDL/M [7]. A PDDL/M domain may contain an additional section that declares the modules similar to the way predicates are declared in PDDL. Declarations start with a unique identifier to reference the module including a possibly empty list of parameters, similar to a function or predicate entry in their respective sections in PDDL. Only for *effect-applicator* modules we then list any number of numerical fluents that are set by the module. Both types of modules then declare the type and finally the function and library name where the module can be found by the planning system.

A *condition-checker* module used in the *manipulation* domain is declared as follows:

```
(:modules
  (checkTransit ?target - movable
    ?place - static ?grasp - grasp
    conditionchecker checkTransit@libTraj.so))
```

This module is called *checkTransit*. It decides whether it is possible for the manipulator to grasp the movable object $?target$, located at $?place$ using grasp $?grasp$, and can be found in the shared library `libTraj.so` by calling the function `checkTransit`.

The syntax of *effect-applicator* modules is similar, as can be seen in the following excerpt:

```
(:modules
  (applyTransit ?target - movable
    ?place - static ?grasp - grasp
    (q0) (q1) (q2) (q3) (q4) (q5) (q6)
    (p0 ?target) (p1 ?target) (p2 ?target)
    (p3 ?target) (p4 ?target) (p5 ?target)
    (p6 ?target) (p7 ?target) (p8 ?target)
    (p9 ?target) (p10 ?target) (p11 ?target)
    effect applyTransit@libTraj.so))
```

This module sets the resulting seven DOF manipulator configuration $(q0)$ - $(q6)$ and the target's new transformation matrix $(p0\,?target)$ - $(p11\,?target)$ resulting from grasping $?target$ using $?grasp$.

To use a module in an operator, it has to be specified in the same way as predicates or functions. The only new syntax we introduce is that a module is given by enclosing its identifier and parameters in square brackets. All other identifiers used in the following *pick-up* operator follow standard PDDL syntax.

```
(:durative-action pick-up
  :parameters (?x - movable
     ?y - static ?g - grasp)
  :duration (= ?duration 1)
  :condition (and
     (at start (not (arm_moving)))
     (at start (on ?x ?y))
     (at start (handempty))
     (at start ([checkTransit ?x ?y ?g])))
  :effect
    (and
      (at start (arm_moving))
      (at end (not(arm_moving)))
      (at end (not (on ?x ?y)))
      (at end (not (handempty)))
      (at end (holding ?x ?g))
```

```
(at end ([applyTransit ?x ?y ?g])))
```

The implementation of PDDL/M in forward-chaining planners is described in detail in our previous work [7].

III. Semantic attachments for planning transfer and transit paths

Semantic attachments for the robot manipulation domain are implemented as probabilistic roadmap planners (PRM). Upon a call to a condition checker module, the procedure is provided with the operator's parameters: a target object, a place to grasp an object at for transit paths or to put the object to for transfer paths and a grasp to use. The first step is to invoke the provided callbacks to the symbolic planner and thus retrieve the current robot configuration and the object's locations. Based on this information a geometric initial state for the PRM planner is built. To form the goal state, the previously computed initial state is now updated placing the manipulator (and for transfer paths the object) in their desired target positions.

Next, the trajectory planner is called with those computed initial and goal states. The planner computes a roadmap: a graph representing the manipulator's collision free configuration space (C_{free}). The roadmap's nodes are computed by randomly sampling configurations in the robot's configuration space and only retaining collision free samples (i.e. those with a distance bound greater zero). Edges represent collision free paths between nodes in the roadmap. The robot movement that an edge represents is a straight line in C_{free}. To connect two nodes and thus form an edge that is guaranteed to be without collision during the robot's movement performing only collision tests is not sufficient. Therefore, we use proximity queries similar to the method by Schwarzer et al. [6]. Once a roadmap has been built, the initial and goal state are inserted into the roadmap and we try to connect those nodes to the roadmap in the same way as sampled nodes are connected. A simple breadth first search now gives us a path through the roadmap from the initial to the goal state or results in failure if the init and goal nodes are not in the same components of the graph. Successful planning using the trajectory planner will result in a true evaluation of a semantic attachment.

An effect applicator module needs to supply the symbolic planner with the robot configuration and object location resulting from an action. We could again run a probabilistic roadmap planner to generate the geometric plan, but this plan should already have been generated during the call to the condition checker module in the same operator. So, for efficiency reasons, we cache results during condition checker computation and just return those. Another reason for caching results is that due to the random sampling in the roadmap planner, results are not necessarily reproducible.

IV. Proximity queries

Validation of collision free edges in the roadmap requires fast computation of distances. Proximity queries return the minimum separation distance between a pair of arbitrarily shaped non-convex objects in work space. The objects are given as closed non-convex triangulated surface meshes in three-dimensional space. The surface meshes consist of points, edges and triangles.

The algorithm proceeds recursively and can be divided into three stages. The first stage employs a variation of the Gilbert-Johnson-Keerthi algorithm (GJK) [8]. It determines the separation distance between the convex hulls of a pair of non-convex objects. Thus, the result is a lower distance bound for the exact separation distance. An upper distance bound is also derived from the data gathered with GJK. If the lower distance bound is greater than zero, i.e. the convex hulls of the two objects do not overlap, the algorithm proceeds with stage two. The second stage employs spatial hashing [9]. The cell size of the hash grid is determined using the distance bounds found in the first stage. Thus, only primitives within the same cell can still contribute to the exact minimum distance. All other primitive pairs are efficiently culled away by the intrinsic properties of the subdivision scheme.

If the convex hulls of the mesh pair overlap, the algorithm proceeds with stage three. In this stage, information computed by GJK is utilized to adaptively decompose the meshes into sub-meshes and pair-wise repeat the process in stage one recursively. The overall minimum distance between the object pair is the minimum of the set of distances computed for all the sub-mesh pairs. Figure 3 depicts the results of the proximity queries posed for all possible pairs of objects in a environment as green lines.

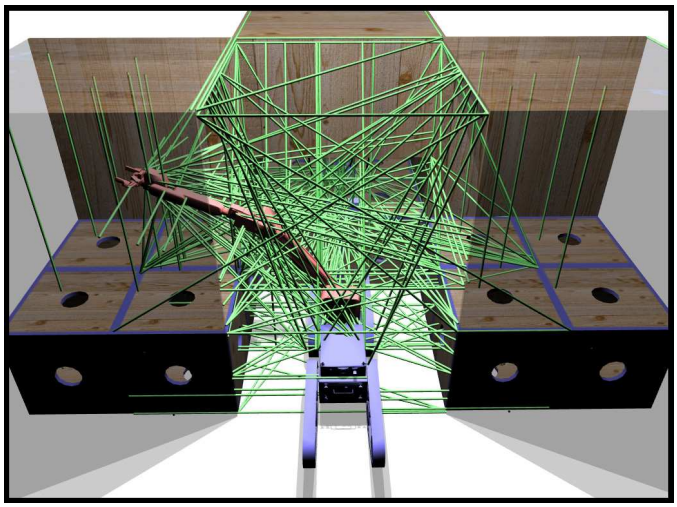

Fig. 3. Proximity queries are posed for all pairs of objects in this test environment. The separation distances are depicted as green lines. The wall and ceiling of the left and right structure are rendered transparent for a better illustration. A vertical board is mounted covering the upper part of the rear compartment.

The algorithm quickly converges to the correct solution. In contrast to other approaches, no offline pre-computations are performed and no acceleration data structures have to be built. A decomposition of surface meshes is only performed, if it is required in the computation of the exact solution. For an in-depth description of the approach, we refer the reader to our previous work [10], [11].

TABLE I

RESULTS FOR THE TABLES SCENE. WE SEPARATED THE PROBLEM INSTANCES IN THREE CLASSES: SIMPLE PICK-AND-PLACE TASKS (CLASS I), PROBLEMS THAT REQUIRE REPLACING ANOTHER OBJECT TO REACH THE GOAL CONFIGURATION (CLASS II), AND PROBLEMS THAT REQUIRE REPLACING MULTIPLE OBJECTS (CLASS III).

Class I	Runtime [s]
01	3.48 ± 1.23
02	6.08 ± 3.49
03	3.44 ± 1.61
04	1.47 ± 0.12
05	3.77 ± 0.97
06	3.98 ± 3.01
07	4.75 ± 2.36
08	5.27 ± 2.71
09	63.83 ± 7.67
10	5.66 ± 7.50
11	12.48 ± 14.74
12	3.30 ± 0.96
13	5.80 ± 2.40

Class II	Runtime [s]
01	24.32 ± 8.63
02	24.95 ± 9.25
03	91.87 ± 14.01
04	30.26 ± 9.74

Class III	Runtime [s]
01	37.33 ± 6.85
02	15.50 ± 2.52
03	78.55 ± 45.61

V. EXPERIMENTS

We evaluate our manipulation planning system by conducting several experiments of increasing difficulty in two environments (see figures 4 and 1). The first environment consists of the robot surrounded by three tables. Various manipulable items are placed on the tables such as bottles or cereal boxes. The second scenario is a reconstruction of a test environment used during the response robot evaluation in Disaster City [2]. Cubes of 60 cm in size and a hole of 15 cm in diameter per side are arranged around the robot. Manipulable cubes of about 8 cm in size are stacked on top. The object representation in these environments is twofold. First, the objects' surfaces are represented as triangular meshes. They provide the input for the proximity query algorithm described in section IV. Second, tetrahedral meshes are used to approximate the objects' volumes and to simulate the physical behavior of movable objects in the world. The robot representation consists of 2400 triangles and 2500 tetrahedrons, respectively. Triangles and tetrahedrons sum up to 2500 and 2600 in the first, and 8000 and 6000 in the second environment, respectively. All runtimes were computed as average runtimes on a Intel Core2Duo E6400 with 2 GB RAM in 32-bit Linux. Although the roadmap creation phase could be parallelized, we only used one core. A video of two exemplary plans can be found at: `http://www.informatik.uni-freiburg.de/~dornhege/media/symbolicManipulationPlanning.avi`.

The two scenarios we used present two different problems. In the tables scene, we formulate problems that place objects at other objects' locations forcing the planner to detect such situations and plan for them accordingly. Results shown in table I indicate that even multiple replacing of objects still results in reasonable runtimes. The manipulation dexterity test scenario usually only contains simple pick-and-place operations grasping the cubes and placing them over the target holes. Its difficulty lies in the fact that a vertical board can be mounted (see figure 3) to highly limit access to the objects. To give comparative results, we evaluated all problem instances with and without the board present. Results in table II show that especially the problem instances requiring to grasp the rear cubes (3 - 5, 10, 11) can be solved quite fast when there is no board obstructing the way (see figure 1 for the cube locations). The most difficult problem is instance 11 that places the two rear cubes in holes at the left and right compartment, so that the manipulator has to be moved from front to back of the vertical board and vice versa four times.

TABLE II

RUNTIMES IN SECONDS FOR THE MANIPULATION DEXTERITY SCENARIO. ALL PROBLEM INSTANCES HAVE BEEN EVALUATED WITH AND WITHOUT THE VERTICAL BOARD PRESENT.

Problem	without board [s]	with board [s]
01	0.06 ± 0.01	0.06 ± 0.01
02	0.06 ± 0.00	0.06 ± 0.00
03	0.17 ± 0.01	59.46 ± 41.92
04	0.17 ± 0.00	67.96 ± 46.87
05	11.22 ± 9.50	207.66 ± 143.61
06	0.12 ± 0.01	0.12 ± 0.00
07	0.39 ± 0.01	0.12 ± 0.00
08	0.23 ± 0.00	0.24 ± 0.01
09	0.23 ± 0.01	0.24 ± 0.00
10	1.51 ± 0.01	162.00 ± 52.99
11	54.79 ± 21.00	978.35 ± 1105.81

VI. RELATED WORK

A. Symbolic planning

Domain-dependent planning systems such as SHOP2 [12], TLPlan [13], or TALplanner [14] are related to our approach as they allow specifying control rules based on domain knowledge. However, the mentioned systems put their effort into allowing the user to specify means how to solve a given symbolic planning problem. In other words, they stay in their symbolic domain, but try to optimize search.

We, however, try to decompose the planning problem into different sub-problems that can be solved independently, but still have non-trivial interactions. In so far, it is similar to the work by Fox and Long [15], who tried to isolate optimization problems from planning problems. Furthermore, the work by Srivastava and Kambhampati on decomposing a general planning problem into a resource and a planning problem [16] is relevant here. However, they investigate how resource and planning problems are related to each other, while we use a general framework for combining different kinds of planning.

The mechanism we use is similar to an undocumented feature of TLPlan [13]. This planner also permits semantic attachments to predicate symbols [17]. The main differences to our approach are that TLPlan is a domain-dependent planner, that one cannot inspect the state the planner is in using call-back functions, and that it is not possible to specify externally computed effects.

B. Manipulation planning

Solving the robotic planning problems in high-dimensional configuration spaces is often addressed using probabilistic

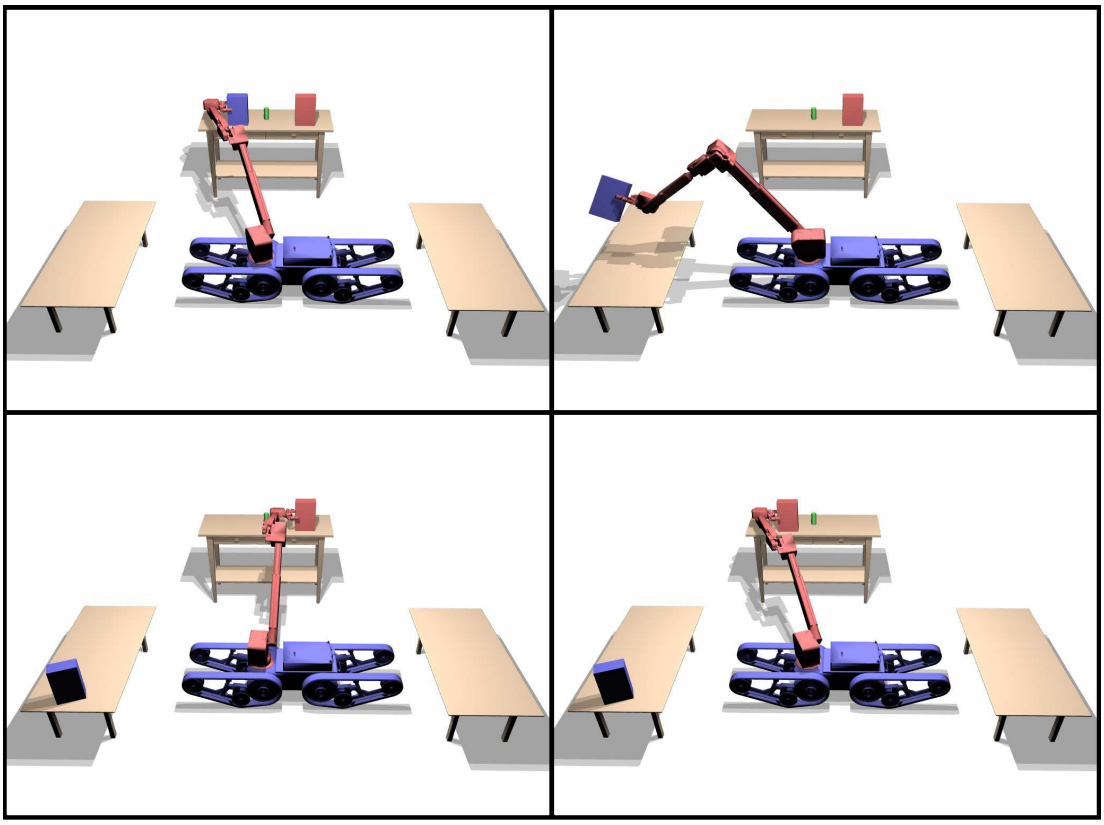

Fig. 4. Execution of a manipulation plan in test environment 1. The manipulator (red) executes the task of placing the red box to where the blue box is located (lower right). Therefore, it first has to remove the blue box from that position (upper left) and place it somewhere else (upper right and lower left). This problem is solved by the symbolic planner and included in the final execution plan.

roadmap planners (PRM) [18], [5], [19]. We also follow this approach when implementing our semantic attachments. The integration of proximity queries in the PRM framework was proposed by Schwarzer et al. [6] allowing to compute proven collision-free trajectories.

Manipulation planning is addressed by building the "manipulation graph" that consists of nodes representing viable grasps and placements. Nodes are connected by transit or transfer paths moving either the manipulator alone or together with a grasped object. Those paths are solved using PRM planners [4], [20].

The work that comes closest to our intentions in the area of robotic planning is the work by Cambon et al. [21], [22]. They also work on the integration of manipulation and symbolic planning. However, in contrast to our work, they did not try to identify a general interface between symbolic planning and domain planning, but presented a specialized combination of a symbolic and a manipulation planner.

C. Proximity queries

Proximity query algorithms can be classified into three categories: collision detection, separation distance computation and penetration depth computation. Generally, the first two categories are of interest in the context of motion planning. Over the last decades, a large variety of proximity query algorithms has been proposed. Many algorithms exploit the properties of convex sets to be able to formulate a linear programming problem. Queries for separation distance [8], [23], collision [24] or penetration depth queries [25] can, thus, be answered efficiently. In dynamic environments, geometric and time coherence can be exploited to track the closest points [23], [25]. These algorithms can be employed on non-convex sets, if the sets are either considered as compositions of several convex subsets [8], [23], or non-convex sets are decomposed into convex subsets [26]. The algorithms are then applied to the convex subsets, respectively. To accelerate the pairwise proximity query, the sets can be stored in bounding volume hierarchies. Different types of bounding volumes have been investigated [27], [28], [29], [30]. In terms of collision detection, spatial subdivision schemes are employed to rule out pairs of sets that are not spatially coherent [9]. Graphics hardware can be used to accelerate various geometric computations such as collision detection [31], [32], or distance field computation [33], [34]. Possible drawbacks of GPU-based approaches are their accuracy due to frame buffer resolution or the read-back time of frame buffers to the CPU memory. A hybrid approach that combines the efficiency of a distance computation approach for convex objects and the benefits of a spatial subdivision scheme is proposed in [10] and extended in [11]. For a more detailed discussion about proximity queries, excellent surveys can be found in [35]

and [36].

VII. Conclusion

We presented a solution to the robotic manipulation planning problem. By tightly integrating symbolic and geometric planning we gained a well performing system that furthermore allows to formulate goals in an intuitive symbolic manner as "put the box on the table" resulting in collision free trajectories even in complex scenarios. The runtimes of this initial implementation are already viable for most scenarios, although we believe that the most complex problems still need improvement. This is one of the tasks that we will address in the future. We plan on integrating geometric heuristics in the symbolic planning process to significantly reduce calculation times. We will also work on accurate world modelling using laser range finders that are mounted on our robot.

Acknowledgment

This research was supported by DFG as part of the collaborative research center SFB/TR-8 Spatial Cognition Project R7.

References

[1] A. Jacoff and E. Messina, "Urban search and rescue robot performance standards: Progress update," in *SPIE Defense and Security Conference*, 2007.

[2] T. Engineering Extension Service, "TEEX Disaster City," http://www.teex.com/teex.cfm?templateid=1117, June 2009.

[3] R. W. Weyhrauch, "Prolegomena to a theory of mechanized formal reasoning," *Artif. Intell.*, vol. 13, no. 1-2, pp. 133–170, 1980.

[4] R. Alami, J. P. Laumond, and T. Siméon, "Two manipulation planning algorithms," in *WAFR: Proceedings of the workshop on Algorithmic foundations of robotics*. Natick, MA, USA: A. K. Peters, Ltd., 1995, pp. 109–125.

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

[6] F. Schwarzer, M. Saha, and J. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Transactions on Robotics and Automation*, vol. 21(3), pp. 338–353, 2005.

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

[8] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Transactions on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[9] M. Teschner., B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *Vision, Modeling, Visualization VMV'03, Munich, Germany*, 2003, pp. 47 – 54.

[10] M. Gissler, U. Frese, and M. Teschner, "Exact distance computation for deformable objects," in *Proc. Computer Animation and Social Agents CASA'08*, 2008, pp. 47–54.

[11] M. Gissler and M. Teschner, "Adaptive surface decomposition for the distance computation of arbitrarily shaped objects," in *Proc. Vision, Modeling, Visualization VMV'08*, 2008, pp. 139–148.

[12] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wau, and F. Yaman, "Shop2: An HTN planning system," *JAIR*, vol. 20, pp. 379–404, 2003.

[13] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artif. Intell.*, vol. 116, no. 1–2, pp. 123–191, 2000.

[14] J. Kvarnström and P. Doherty, "TALplanner: A temporal logic based forward chaining planner," *Ann. Math. Artif. Intell.*, vol. 30, no. 1-4, pp. 119–169, 2000.

[15] M. Fox and D. Long, "Identifying and managing combinatorial optimisation subproblems in planning," in *Proc. IJCAI*, 2001, pp. 445–452.

[16] B. Srivastava and S. Kambhampati, "Scaling up planning by teasing out resource scheduling," in *Proc. ECP*, 1999, pp. 172–186.

[17] A. Botea, M. Müller, and J. Schaeffer, "Using abstraction for planning in sokoban," in *Proc. Computers and Games*, Edmonton, Canada, 2003, pp. 360–375.

[18] J. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[19] G. Sanchez and J. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *Springer Tracts in Advanced Robotics*, vol. 6, pp. 403–417, 2003, published in: Robotics Research: The Tenth Int. Symp.

[20] T. Simeon, J. Cortes, J. Laumond, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[21] S. Cambon, F. Gravot, and R. Alami, "A robot task planer that merges symbolic and geometric reasoning," in *Proc. ECAI*. IOS Press, 2004, pp. 895–899.

[22] S. C. Fabien Gravot and R. Alami, "asymov:a planner that deals with intricate symbolic and geometric problems," *Springer Tracts in Advanced Robotics*, vol. 15, pp. 100–110, 2005.

[23] M. Lin and J. Canny, "A fast algorithm for incremental distance calculation," in *IEEE International Conference on Robotics and Automation*, 1991, pp. 1008–1014.

[24] G. van den Bergen, "A fast and robust GJK implementation for collision detection of convex objects," *J. Graphics Tools*, vol. 4, no. 2, pp. 7–25, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=334711

[25] S. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra," *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3112–3117, 1997.

[26] S. Ehmann and M. Lin, "Accurate and fast proximity queries between polyhedra using surface decomposition," *Computer Graphics Forum (Proc. of Eurographics'2001)*, vol. 20, no. 3, pp. 500–510, 2001.

[27] S. Quinlan, "Efficient distance computation between non-convex objects," *IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3324–3329, 1994.

[28] P. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics*, vol. 15, no. 3, pp. 179–210, 1996.

[29] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Tran. on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[30] S. Gottschalk, M. Lin, and D. Manocha, "OBB-Tree: a hierarchical structure for rapid interference detection," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1996, pp. 171–180. [Online]. Available: http://portal.acm.org/citation.cfm?id=237244

[31] D. Knott and D. Pai, "CInDeR: Collision and interference detection in real-time using graphics hardware," in *Proc. of Graphics Interface*, 2003, pp. 73–80.

[32] N. Govindaraju, S. Redon, M. Lin, and D. Manocha, "CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware," in *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Aire-la-Ville, Switzerland: Eurographics Association, 2003, pp. 25–32.

[33] K. Hoff, J. Keyser, M. Lin, and T. Manocha, D. andCulver, "Fast computation of generalized voronoi diagrams using graphics hardware," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1999, pp. 277–286.

[34] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha, "Fast proximity computation among deformable models using discrete Voronoi diagrams," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1144–1153, 2006.

[35] M. C. Lin and D. Manocha, *Handbook of Discrete and Computational Geometry*. CRC Press, 2004, ch. 35, pp. 787 – 806.

[36] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision detection for deformable objects." *Computer Graphics Forum*, vol. 24, no. 1, pp. 61 – 81, 2005.