Journal of Artificial Intelligence Research 61 (2018) ??-?? Submitted 04/17; published ??/18

KABouM: Knowledge-Level Action and Bounding Geometry Motion Planner

Andre Gaschler GASCHLERA@GMAIL.COM
fortiss An-Institut Technische Universität München
Munich, Germany

Ronald P. A. Petrick R.PETRICK@HW.AC.UK
Department of Computer Science, Heriot-Watt University
Edinburgh, United Kingdom

Oussama Khatib KHATIB@STANFORD.EDU
Artificial Intelligence Laboratory, Stanford University
Stanford, USA

Alois Knoll KNOLL@IN.TUM.DE
Institut für Informatik, Technische Universität München
Garching b. München, Germany

Abstract

For robots to solve real world tasks, they often require the ability to reason about both symbolic and geometric knowledge. We present a framework, called KABouM, for integrating knowledge-level task planning and motion planning in a bounding geometry. By representing symbolic information at the knowledge level, we can model incomplete information, sensing actions and information gain; by representing all geometric entities—objects, robots and swept volumes of motions—by sets of convex polyhedra, we can efficiently plan manipulation actions and raise reasoning about geometric predicates, such as collisions, to the symbolic level. At the geometric level, we take advantage of our bounded convex decomposition and swept volume computation with quadratic convergence, and fast collision detection of convex bodies. We evaluate our approach on a wide set of problems using real robots, including tasks with multiple manipulators, sensing and branched plans, and mobile manipulation.

1. Introduction

The ability to reason about action and geometry is a key capability for building intelligent robots that can operate in service, manufacturing, construction, or teleoperation capacities. While modern robot technology has lead to reliable platforms and novel sensors, and enormous progress has been made in providing the necessary set of algorithmic components for path planning, navigation, and perception, many tasks that require complex reasoning about actions and geometry remain problematic, especially in real-world settings.

For humans, the process of reasoning about action and geometry is a central, high-level cognitive function that is required to solve many types of complex, but common, tasks. It is therefore useful, and often necessary, for intelligent robots to reason at similar levels of abstraction, using similar geometric concepts. For instance, many tasks in service and industrial robotics include communication with humans, making it necessary for the robot to understand some human concepts of geometry and action. Second, almost all robot tasks

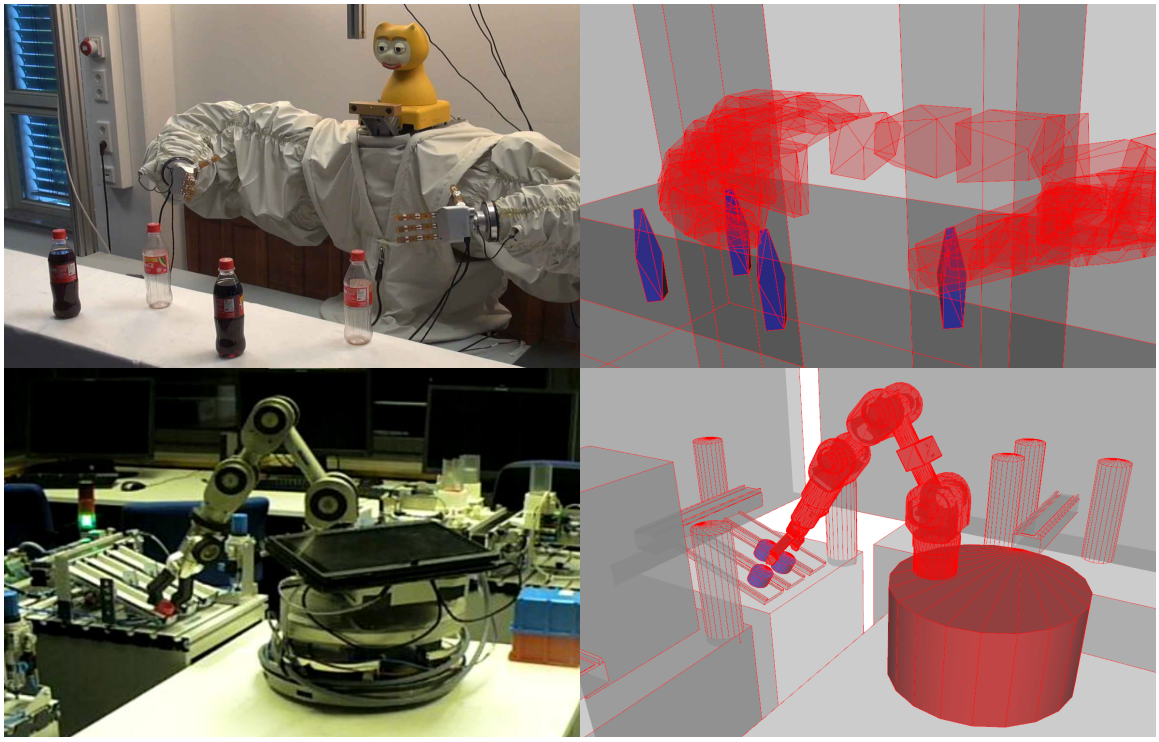


Figure 1: Robot tasks are modelled by symbolic actions and geometric volumes, representing both swept volumes of robot motions (red), and object boundaries as sets of convex polyhedra (blue for movable objects, grey for static obstacles).

take place in a world designed by humans, so robots need to interact and solve tasks with objects and agents using actions that are also similar to those of humans.

Our approach for solving complex robotic task problems involves combining general-purpose AI planning techniques with a problem specification described using a concise symbolic and geometric domain definition. This domain definition is written from a generic task perspective, not anticipating a specific solution. Instead, we utilise our planning framework to generate solutions based on specific task formulations.

However, the problem of combining task planning and motion planning in a robot system presents significant representational difficulties that must be overcome: high-level task planners typically rely on symbolic representations of objects and actions, while motion planning systems need to reason about physical bodies and robot motion in a continuous space. Integrating symbolic and geometric reasoning in a common framework is therefore both a challenging and essential task.

For instance, Figure 1 (top) shows an example of a service robot that can serve drinks, ask customers for their orders, move so it can reach objects, and transfer those objects to customers (Foster, Gaschler, Giuliani, Isard, Pateraki, & Petrick, 2012). For even this simple scenario, the task planner already needs to generate information-gathering actions, evaluate geometric preconditions, and fulfil geometric constraints such as holding a drink upright while moving.

In this paper, we propose a task planning framework called KABouM (Knowledge-level Action and Bounding geometry Motion planner) that addresses many of these challenges. Key to this approach is a series of geometric calculations that operate on *single-sided approximations* of the geometry. Since robotic tasks have asymmetric tolerances, collisions must be strictly avoided for they can cause damage or other unforeseen effects. We must therefore detect all types of collisions, quickly and reliably. Single-sided approximations enable us to do so with increased efficiency. Contrary to overly coarse bounding boxes or convex hull approximations, which severely limit the search space, we design our approximation as *ε-precise*, meaning we may only overlook passages thinner than a parameter ε, which has a controllable effect on the search space. We realise this approach by introducing the bounding mesh algorithm, which approximates world geometry by coarser triangle meshes that fully enclose the given shapes. Compared to regular, two-sided approximation in computer graphics (Mamou & Ghorbel, 2009) and robotics (Schulman, Lee, Awwal, Bradlow, & Abbeel, 2013), we develop a single-sided solution (Gaschler, Fischer, & Knoll, 2015; Gaschler, 2016) that guarantees collision avoidance. Formally, we define a class of *bounded ε-precise geometric predicates* for collision and inclusion, which operate on bounding convex decompositions.

High-level symbolic planning in this framework is performed by using an off-the-shelf, general-purpose planner which is integrated with the geometric representation. In particular, we use the existing Planning with Knowledge and Sensing (PKS) planner (Petrick & Bacchus, 2002, 2004) that is able to reason with incomplete information (discrete uncertainty), which is common in real-world robotics problems, and that supports contingent planning and sensing actions. PKS has been previously used to connect high-level planning with robot systems (Kraft, Baseski, Popović, Batog, Kjær-Nielsen, Krüger, Petrick, Geib, Pugeault, Steedman, Asfour, Dillmann, Kalkan, Wörgötter, Hommel, Detry, & Piater, 2008; Petrick, Kraft, Krüger, & Steedman, 2009; Petrick & Foster, 2013), and has a comprehensive application programming interface (API) supporting easy integration (Petrick, 2015).

The remainder of the paper is organised as follows. In Section 2, we situate our work with respect to related approaches. In Section 3, we introduce our robot task planning problem. In Section 4, we propose a set of algorithms for evaluating bounded geometric predicates, namely convex decomposition and quadratically converging swept volume approximation. In Section 5, we outline our approach from the symbolic perspective. Finally, we describe the system architecture in Section 6, and evaluate our approach with several illustrative scenarios in Section 7. We discuss our approach and results in Section 8 and conclude in Section 9.

2. Related Work

We are motivated by the problem of applying automated planning techniques to robot systems to enable them to solve more complex problems in industrial and service robotics. Although automated planning has already been applied to robot problems in the early seminal works (Fikes & Nilsson, 1971), and it has been used for robot manipulation since the days of early robotic systems like Shakey (Nilsson, 1984), more recently the field has gained substantial attention, both in the automated planning and robotic motion planning communities. Today's approaches to robot task planning include a diverse range of techniques

which could be categorised with respect to their search strategy, their field of application, or related properties such as execution and monitoring.

2.1 General Approaches to Robot Task Planning

Planning algorithms include a diverse range of different search strategies from closed-world symbolic planning guiding motion planning (Cambon, Alami, & Gravot, 2009; Dornhege, Gissler, Teschner, & Nebel, 2009b; Karlsson, Bidot, Lagriffoul, Saffiotti, Hillenbrand, & Schmidt, 2012), multi-modal motion planning (Hauser & Ng-Thow-Hing, 2011; Barry, 2013), probabilistic back-chaining (Kaelbling & Lozano-Pérez, 2013), and formal synthesis (Kress-Gazit & Pappas, 2008; Cheng, Geisinger, Ruess, Buckl, & Knoll, 2012).

One underlying challenge of robot task planning is the hybrid search space of both continuous-space geometry about robot motion and perception, as well as the discrete symbolic constraints needed to represent knowledge about actions and objects. Approaches for addressing the problem of hybrid search vary greatly in the literature, with some strategies clearly guided by symbolic reasoning (Karlsson et al., 2012; Dornhege et al., 2009b; Cheng et al., 2012), and others by motion planning (Hauser & Ng-Thow-Hing, 2011; Barry, 2013). The direction of the search itself may be a forward search starting from an initial state (Plaku & Hager, 2010; Srivastava, Riano, Russell, & Abbeel, 2013; Gaschler, Petrick, Giuliani, Rickert, & Knoll, 2013a), a backward search directed from a goal (Kaelbling & Lozano-Pérez, 2013), or both (Barry, 2013; Garrett, Lozano-Perez, & Kaelbling, 2015). Another notable property of a hybrid planner is whether backtracking passes geometric knowledge to the symbolic level (Srivastava et al., 2013; Kaelbling & Lozano-Pérez, 2013; Cambon et al., 2009) or high-level symbolic actions are mostly refined by motion planning (Erdem, Haspalamutgil, Palaz, Patoglu, & Uras, 2011; Dearden & Burbridge, 2013). Finally, some approaches interleave planning with direct execution after each planned step (Haigh & Veloso, 1998; Kaelbling & Lozano-Pérez, 2012), which greatly reduces the search space, especially when actions are reversible, while others focus their geometric search using structures like semantic maps (Galindo, Fernández-Madrigal, González, & Saffiotti, 2008).

Applications have also provided a driving force for the integration of automated planning and robotics, including scenarios for controlling autonomous robots in remote environments out of the reach of human operators (Bellingham & Rajan, 2007), situations requiring provably correct behaviour (Cheng et al., 2012), and the need to control the complex kinematics of humanoids (Hauser & Ng-Thow-Hing, 2011). However, most of the recent work in robot task planning is motivated by examples in mobile manipulation (Kaelbling & Lozano-Pérez, 2012; Cambon et al., 2009; Wolfe, Marthi, & Russell, 2010; Srivastava et al., 2013), some of which involve service tasks and interaction with humans (de Silva, Pandey, & Alami, 2013).

2.2 Symbolic Planners with Geometric Queries

Dornhege et al. (2009b) integrate geometric reasoning into a symbolic planner using the idea of semantic attachments, which they develop as an extension to the planning domain definition language (Dornhege et al., 2009a). Their interface includes condition-checking semantic attachments, which call a binary robotics-specific function as part of a symbolic precondition, and effect applicators, which pass domain-specific return values to the symbolic planner. Using a probabilistic roadmap motion planner, they apply their approach

in pick-and-place scenarios. Dornhege et al's (2009b, 2009a) external function, or semantic attachment, mechanism is kept generic enabling it to be interfaced to a range of robotics-specific components.

A related approach by Gregory, Long, Fox, and Beck (2012) describes Planning Modulo Theories (PMT) which propose a new planning definition language that introduces abstract data types and subroutines similar to semantic attachments to resolve fluent values. Hertle, Dornhege, Keller, and Nebel (2012) describe the Object-oriented Planning Language (OPL) as an object-oriented way to define planning domains with external modules, also building on Dornhege et al's (2009b) approach. Bajada (2016) proposes an approach called PDDLx where module predicates, functions, and methods provide the semantic attachment necessary to perform computation by an external class module.

Cambon et al. (2009) perform an interleaved search with backtracking in both the symbolic and geometric layers, with the symbolic planner acting as a heuristic for a probabilistic roadmap motion planner. Their integrated approach is similar to Dornhege et al's (2009b), with a generic interface between the symbolic and robotics-specific planners.

Erdem et al. (2011) consider another similar approach that combines high-level reasoning with geometric reasoning and motion planning, in this case using a causality-based formalism. In particular, a causal reasoner guides the motion planner by finding an appropriate task plan, whose generation may include reasoning about external predicates linked to geometric models and kinematic relations. The interface between causal reasoning and geometric reasoning is flexible, and can be used to transfer feasibility checks to external predicates or make domain-level modifications to planning problems as a result of motion planning failures.

Within the GeRT project,[1] which focuses on manipulation tasks, Karlsson et al. (2012) apply a hierarchical task network (HTN) planner and geometric suggester functions to generate geometric states, demonstrated on a humanoid manipulator, while Dearden and Burbridge (2013) map symbolic predicates to geometric probability distributions. Hierarchical task networks have been formalized by Erol, Hendler, and Nau (1994) and have also been previously used to solve non-deterministic problems (Kuter et al., 2009).

The state-abstracted hierarchical task network (SAHTN) planner by Wolfe et al. (2010) uses an integration of symbolic planning with sampling-based kinematic optimisation. Notably, their planner builds a global, state-abstracted cache that is keyed by actions and all state variable values relevant to that action. Later queries can then merge these cached results with irrelevant variables and become more efficient without sacrificing optimality.

Srivastava et al. (2013) propose a generic transformation of continuous-valued parameters into discrete symbols. Their symbolic planner initially passes a solution of the purely discrete problem to a robotics-specific motion planner, which then returns a parameter assignment that violates geometric preconditions, allowing more refined discrete planning, until the entire problem is solved. In comparison to Cambon et al.'s (2009) earlier work, where backtracking is performed in both the discrete and continuous layers, Srivastava et al. (2013) appear to provide a more generic translation applicable to different types of action preconditions and parameters. Srivastava, Fang, Lorenzo, Chitnis, Russell, and Abbeel (2014) extend their approach to hybrid planning with symbolic explanations for geometric

1. `http://www.gert-project.eu/`

failures. They can prove that their search is complete for pick-and-place and related domains. For domains that are restricted to rigid manipulation, Garrett et al. (2015) provide a more efficient backward-forward search.

While most of the above planners (Wolfe et al., 2010; Karlsson et al., 2012) generate complete low-level plans, the hierarchical planner "in the now" (HPN) by Kaelbling and Lozano-Pérez (2011) builds only an abstract plan, and then recursively executes and refines this plan. They show that this interleaved planning and real-world execution strategy is very efficient in non-deterministic domains, under a weak constraint that plan steps are serialisable. At the same time, HPN operates in continuous geometry, with geometric predicates and suggesters that do not rely on discretisation.

Other approaches like the one by Ferrer-Mestres, Frances, and Geffner (2015) have also explored the use of Functional STRIPS combined with compilation approaches to convert geometric task planning problems into standard classical planning problems, solvable with existing off-the-shelf planners.

2.3 Planning in Belief Space

In more recent work, Kaelbling and Lozano-Pérez (2013) describe their belief-space hierarchical planner "in the now" (BHPN), which integrates several interesting approaches for modelling probability distributions over states and reasoning about probability-based uncertainties. In contrast to their belief space approach, the Planning with Knowledge and Sensing (PKS) planner (Petrick & Bacchus, 2002, 2004) builds plans by reasoning directly about the knowledge-level effects of actions of plans with discrete uncertainties (i.e., without probabilities), and has previously been used to connect robot vision and grasping with symbolic action (Petrick et al., 2009). Both the belief space and knowledge-level approaches allow sensing and information-gathering actions to be modelled, arising from the preconditions of manipulation actions, without the need to be hard-coded as tasks themselves. However, rather than attempting to model arbitrary belief states, PKS provides a restricted representation of knowledge with limited reasoning. This gives rise to a compact state representation with an efficient plan generation procedure. Moreover, unlike BHPN, which appears to be tailored to problems in the robotics domain, PKS is a general-purpose planner which is domain independent. Thus, like other automated planners, the same planning approach can be used in a variety of contexts outside or tangential to the robotics domain. The search schemes used by the two approaches are also fundamentally different, as BHPN takes a strongly hierarchical goal regression approach and interleaves planning and execution, while PKS uses forward search in a "flat" action model with simple heuristics to plan over large horizons in a purely offline mode. PKS is an early example of an epistemic planner (Bolander, 2017), a class of planners which attempt to reason and plan with models of knowledge and belief.

When only observations are uncertain and actions are deterministic, Hadfield-Menell, Groshev, Chitnis, and Abbeel (2015) can apply maximum-likelihood estimation to transform problems to purely deterministic representations, which can then be solved by off-the-shelf planners.

2.4 Motion Planning Approaches to Task Planning

Among the approaches that use sampling methods in robot configuration spaces, Hauser and Ng-Thow-Hing's (2011) randomised multimodal motion planning (Random-MMP) strategy combines discrete contact states and continuous configuration of robots and objects in a hybrid state space, and solves a wide range of manipulation and locomotion problems. Random-MMP samples discrete mode switches, such as contact state changes of robots with objects and the environment, and allows manipulation planning with complex kinematic and dynamic constraints.

Barry (2013) presents the diverse action manipulation rapidly-exploring random tree (DARRT) planner. It samples in a Cartesian space of robot and object poses, and performs a bidirectional search, using projection functions to cross lower-dimensional subspaces between states. With this, complex mobile manipulation tasks can be solved, including diverse, non-prehensile actions, and some use of tools. Barry (2013) states that, while DARRT builds on the multi-modal motion planning idea (Hauser & Ng-Thow-Hing, 2011), the mode samplers implemented in DARRT are less problem-specific than those in Random-MMP.

Plaku and Hager (2010) present the sampling-based motion and symbolic action planner (SMAP), which allows differential motion constraints. Their search dynamically samples feasible robot motions and explores regions suggested by a symbolic planner, which accepts task definitions in a STRIPS format. The interaction of continuous and symbolic layers is guided by a utility function heuristic.

3. Robot Task Planning

Integrated robot task planning is a multidisciplinary problem, requiring solutions ranging from formal methods to robot motion planning. KABouM (Knowledge-level Action and Bounding geometry Motion planner) combines several such techniques which we discuss below. To help situate our work, we begin by introducing the general task planning problem, which brings together the two main influences on this work: automated planning and geometric reasoning. We follow this with an overview of our motivating examples, and a comprehensive discussion of the central components that make up KABouM.

3.1 The Robot Task Planning Problem

We begin by defining the underlying problem that our work aims to solve, namely the *robot task planning problem*. Formally, a robot task planning problem instance is defined as a 3-tuple (Σ, R, M), where Σ is a symbolic planning domain defining a set of planning actions, an initial state, and a set of goals; R is the set of kinematics definitions for all robots; and M is the set of geometric models for all robots and objects in the domain. Each of these components can also be decomposed into a number of more basic structures, which we outline below.

The symbolic domain Σ is a tuple $(\mathcal{S}, \mathcal{A}, I, G)$, where $\mathcal{S}$ is a set of symbols defining the vocabulary for the discrete domain, $\mathcal{A}$ is a set of actions, I is a description of the initial discrete state, and G is a set of goal conditions. Actions in a robot task planning domain may contain several kinematic and geometric predicates—such as reachability, collision, and inclusion—which may be included when specifying preconditions, effects, and goal

conditions. Each action $\alpha \in \mathcal{A}$ is itself a 3-tuple of the form $(\vec{x}, pre(\alpha), \mathit{eff}(\alpha))$, where $\vec{x}$ are α's parameters, $pre(\alpha)$ are α's preconditions, and $\mathit{eff}(\alpha)$ are α's effects. The complete formal definition of $\mathcal{A}$ will be given in Section 5.

The set of kinematic chains R contains one element for each robot in the domain, and each kinematic chain is given as a tuple (s, FK, q_0), with the associated discrete symbol $s \in \mathcal{S}$, the forward kinematic function FK and the initial configuration q_0. q_0 is an n-dimensional vector, with its components representing initial angles of revolute joints and lengths of prismatic joints. The forward kinematic function FK is a mapping $\mathbb{R}^n \times \{0, 1, \ldots, n\} \mapsto \mathbb{R}^3 \times \mathbb{SO}(3)$ from the configuration and rigid body index to the spatial position of that rigid body of the robot; if no rigid body is specified, the tool centre of the robot is assumed. FK is typically only defined for a configuration space part of $\mathbb{R}^n$, depending on the physical joint boundaries.

Finally, the geometric world is given by a set of geometric models M. Each geometric model is a tuple $(s, \mathcal{B}, x_0)$, with the associated discrete symbol $s \in \mathcal{S}$, the initial spatial pose x_0 and a set of geometric meshes $\mathcal{B}$ for its rigid bodies. Each geometric mesh in $\mathcal{B}$ may be given as a set of triangle tuples (p_1, p_2, p_3) of 3D vertex positions, defining the boundary of a rigid body, with $(p_2 - p_1) \times (p_3 - p_1)$ pointing outside. (Most objects have only a single rigid body, while for serial kinematic chains, $|\mathcal{B}| = n + 1$.) More details about the geometric algorithms are given in Section 4.

Thus, given a problem instance (Σ, R, M), the general robot task planning problem is to generate a sequence or tree of actions whose execution in the initial state (I, q_0, x_0) brings about a state in which the goal conditions G are satisfied. All actions in this plan must (under ideal conditions) satisfy their respective preconditions and produce their expected effects, with many of these preconditions and effects depending on the kinematic and geometric structures R and M.

3.2 Example KABouM Domains

In order to motivate our work and illustrate our KABouM approach, we will refer to a series of example domains, some of which are used as running examples throughout the paper. In the main discussion, we present the Bartender domain, where a robot with two arms must clean up all empty bottles from a table and move them to a certain "dishwasher" area (Gaschler et al., 2013a). The robot can visually detect whether a bottle is empty, and pick up and put down this type of object. To solve this scenario, the robot needs to hand-over objects by placing them in reach of the second arm. In order to show an example of interdependent manipulation and sensing actions, we also include the Force Sensing domain, where a single robot arm is supposed to remove objects from a table depending on their weight (Gaschler, Petrick, Kröger, Knoll, & Khatib, 2013c). The examples that accompany our discussion are kept intentionally simple and are chosen to highlight and evaluate specific features of KABouM. In Section 7, we will additionally define and evaluate more integrated scenarios to discuss performance and scalability of our approach on a quantitative level.

Using our notion of a robot task planning problem and our motivating examples, in the following sections we provide a description of our algorithms for bounded geometric predicates (Section 4), an overview of the knowledge-level PKS planner (Section 5), and the integrated KABouM software architecture (Section 6).

4. Geometric Algorithms

An important aspect of our KABouM approach is to provide efficient geometric predicates. In particular, we represent all geometric entities—boundaries of objects, robots and swept volumes of motions—as sets of convex polyhedra, yielding a representation to bridge the gap between discrete symbols and continuous-valued objects and motions in the world. For this, we derive and implement a completely new class of *bounded geometric predicates*. We begin by introducing a set of robotics-specific predicates from the symbolic planner's perspective, for which we provide the following primitive queries:

Definition 1 (Bounded Geometric Predicates) *The following geometric predicates are provided to the symbolic planner as external procedures:*

1. $m_0 \cap m_1 \neq \emptyset$, *whether two geometric models collide.*
2. $\mathrm{SV}(r_0, Q_0 = \{q_0, q_1, \ldots\}) \cap m_1 \neq \emptyset$, *whether the swept volume of a robot motion collides with an object.*
3. $\mathrm{SV}(r_0, Q_0) \cap \mathrm{SV}(r_1, Q_1) \neq \emptyset$, *whether two robot motions collide.*
4. $m_0 \subset m_1$, *whether a geometric model includes another.*

These collision and inclusion predicates are said to be single-sided ε-precise: their answer is arbitrary when the signed Hausdorff distance between the bodies is within $[0, \varepsilon]$, and exact otherwise. We refer to these single-sided approximate queries as bounded geometric predicates.

We define this novel, bounded and one-sided approximation as follows: a collision query is bounded ε-precise if it reports true when a collision occurs, and false when the Hausdorff distance between the two models is greater than ε. Likewise, an inclusion query "$m_0 \subset m_1$" is bounded ε-precise if it reports false when $m_0 \not\subset m_1$, and true when m_0 is further than ε inside m_1. For distances between 0 and ε, these queries may answer arbitrarily. Intuitively, the Hausdorff distance between two meshes is the maximum of the shortest distances to the other mesh over all points of both meshes. The signed Hausdorff distance is defined negative if the bodies intersect. It is a suitable metric for mesh approximation (Garland & Heckbert, 1997) and gives a limit on the size of narrow passages that can potentially be overlooked in motion planning.

We will later see that this definition of predicates facilitates a very efficient implementation with our novel bounding mesh algorithm, bounded convex decomposition, and swept volume generation. In general, we can always achieve a bounded approximate convex decomposition of all objects, robot models, and swept volumes of robot motions, and only use efficient convex-convex checks in the underlying implementation. From the planning side, it is worth noting that all geometric approximation is controlled by a parameter ε, and our geometric predicates affect the geometric search space by a meaningful distance ε, which may be chosen to be arbitrarily small. ε essentially controls the compromise between more precise or more efficient (but always one-sided approximate) geometric predicates.

We now outline the algorithms necessary to implement the above predicates. In our discussion, we first introduce our new bounding mesh algorithm, which is necessary for

computing bounded convex decompositions. With this, bounded convex decompositions of all robot and object models $\mathcal{B}$ reduce most of the above queries to simple convex-convex collision or inclusion checks, which can be solved efficiently. For swept volumes of robot motions, we furthermore describe a new swept volume generation scheme that exploits a curvature limit of robot paths to keep the number of those convex-convex checks to a minimum. In our implementation of the new *bounded ε-precise* class of geometric predicates, we can only partly rely on available algorithms (Garland & Heckbert, 1997; Mamou & Ghorbel, 2009; Xavier, 2002), and adapt some of these to ensure the boundary constraint.

4.1 Bounding Convex Decomposition of Volumes

In our KABouM approach, it is a central strategy to represent the surfaces of arbitrary objects and robot manipulators by bounding sets of convex polyhedra. A bounding set of convex polyhedra is a union of convex polyhedra that completely includes the original model. In the general case, however, decomposing a non-convex polyhedron into a small (or even minimal) set of convex polyhedra is a challenging problem: exact decomposition produces far too many polyhedra (see Figure 2b), and it is known to be NP-hard (Chazelle et al., 1997). Despite these difficulties, Mamou and Ghorbel (2009) recently proposed an approximate algorithm that is sufficiently efficient and precise for practical instances, with respect to approximation errors as well as the number of decomposed convex polyhedra. Their algorithm shows better approximation results than the first algorithm by Lien and Amato (2007) and other existing solutions. However, we are concerned with bounded geometric queries, where all approximations are one-sided (i.e., they include the original model) and limited by a certain ε. Mamou and Ghorbel's approximate convex decomposition, being designed for computer graphics applications, fulfils neither of these goals; it cannot generate bounding approximations, and it does not guarantee a maximum geometric error.

Obviously, exact convex decomposition alone does not reduce the number of vertices at all (as shown in Figure 2b), and even Mamou and Ghorbel's segmentation will never reduce vertices in convex surface areas, which are prevalent in most objects. Because of this, it is crucial for our approach to combine their decomposition with a surface simplification technique that ensures a bounded approximation, namely our bounding meshes. (Their approach is to discard all but a constant number of vertices from each convex polyhedron in order to handle this issue; however, this decimation will not generate bounded approximations and is not applicable to our geometric predicates.) We therefore adapt their underlying vertex set segmentation such that it constructs a full decomposition into convexes, and reduce the resulting convex polyhedra with our bounding mesh surface simplification, which we derive below.

4.1.1 Bounding Mesh Surface Simplification

Almost all geometric computations can be performed more efficiently when approximations or level-of-detail models are available. While bounding volumes such as bounding boxes, bounding spheres, or convex hulls have long been used for that purpose, we generalise this concept to the generation of bounding meshes, triangular meshes that enclose more detailed meshes. The only previous works in this direction are by Sander et al. (2000), who compute inner and outer bounding meshes within their silhouette clipping algorithm, and

Algorithm 1 Bounded edge contraction point optimisation

Input: Neighbouring planes P of an edge
Output: Edge contraction point v' and its cost

```
cost ← ∞
for all {0,1,2,3}-subset P' = [N^T  −d^T] of neighbouring planes P do
    A ← nullspace(N)
    b ← −N^{−T} d
    x ← −(A^T P'^T P' A)^{−1} A^T P'^T P' b
    v ← Ax + b
    if P^T v ≥ 0 and v^T P^T P v < cost then
        v' ← v
        cost ← v^T P^T P v
    end if
end for
```

Algorithm 2 Bounding mesh generation with iterative edge contraction

Input: Mesh m, tolerance ε
Output: Bounding mesh $m' \supseteq m$

```
for all edge e ∈ m do
    Evaluate edge contraction cost(e, v') from Algorithm 1
    Add cost(e, v') to priority queue
end for
while cost(e, v') < ε^2 do
    Pop best edge e from priority queue
    Remove edge e and adjacent triangles
    Insert new vertex v' and triangles
    Re-calculate costs of changed edges
end while
```

by Platis and Theoharis (2003), who internally use progressive bounding meshes for efficient line–mesh intersection queries. Our algorithm is novel in that it uses a new, geometrically reasonable distance metric that allows ε-precise approximations, and we are the first to apply bounding meshes to more generic problems.

In order to derive our bounding mesh generation, we choose an iterative mesh simplification approach—similar to the well-known, two-sided mesh approximation by Garland and Heckbert (1997)—and search for iterative edge contractions guided by a quadric error measure. In other words, we simplify a given mesh by replacing two adjacent vertices v_1 and v_2 by a single vertex v' in each iteration, with v' chosen outside of the local mesh, and by that generate a simpler mesh that encloses the original. Let $P(v)$ be the set of planes adjacent to a vertex v, each given as a normal vector. For our simplified mesh to be a bounding mesh, i.e. a superset volume, v' obviously has to be "outside", or mathematically, $p^T v' \geq 0$ for all $p \in P(v_1) \cup P(v_2)$. Besides this constraint, we try to minimise the geometric distance ε to the input mesh—this allows us to choose the best edge to contract, and to compute which 3D point v' that edge is contracted to. An efficient way to define the cost

of an edge contraction is the sum of squared distances of v' to all adjacent planes p, which is $\sum_p v'^T p\, p^T\, v'$. Combining minimisation and constraints, our bounded edge contraction solves the optimisation problem

$$\operatorname*{argmin}_{v'} \sum_{p \in P(v_1) \cup P(v_2)} v'^T p\, p^T\, v' \quad \text{s.t. } \forall p: \; p^T v' \geq 0\,. \tag{1}$$

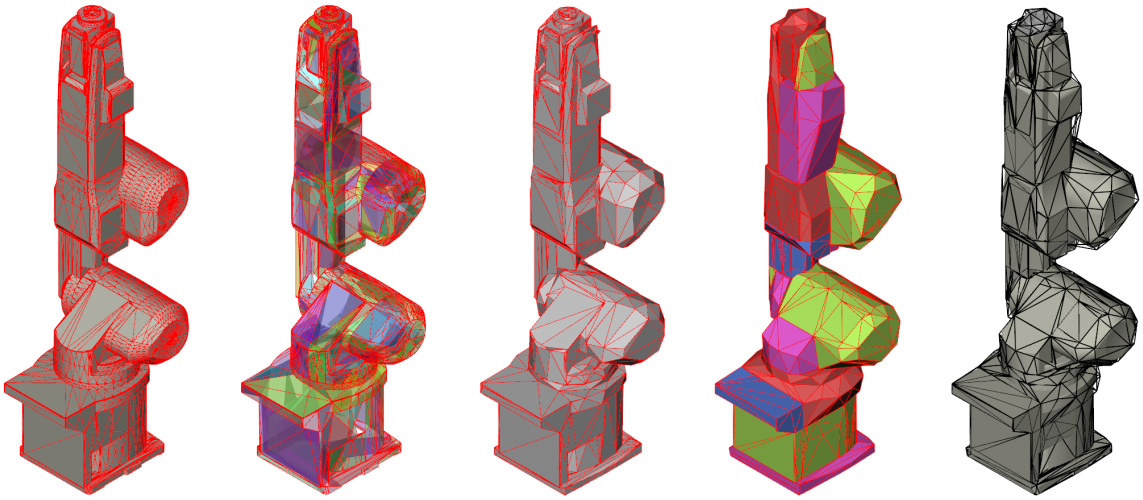
As proposed by Garland and Heckbert (1997), we can simply sum and propagate the quadratic matrix $p\, p^T$ through edge contraction steps. This approximates the cost function in Eq. 1 by simple matrix sums of 4-by-4 matrices, and only slightly affects optimality—costs per plane may be counted twice or three times instead of just once—but not the constraints.

In order to find the minimiser v', we discriminate four cases of Eq. 1: The minimiser is constrained by either zero, one, two, or at least three linear equality constraints. For each case $m \in \{0, 1, 2, 3\}$, we can enumerate all m-subsets of neighbouring planes $P(v_1) \cup P(v_2)$ and exhaustively search the global minimum. The cases $m \in \{0, 1, 2\}$ are solved by substitution into the $(3 - m)$-dimensional constraint subspace and simple unconstrained quadratic optimisation. Case $m = 3$ corresponds to the intersection of three planes, so all four cases reduce to linear systems with up to three dimensions. For each m-subset, we compute the relaxed minimiser v; of those v that fulfil all inequality constraints, we compute the cost function and finally obtain the global minimiser v'. Algorithm 1 shows an implementation of this procedure for solving Eq. 1.

Because the number of neighbouring planes is very limited, our exhaustive search is sufficiently fast, with 10^5 edge decimations per second on our testing system (see Section 7 below). The complete bounding mesh generation is given by Algorithm 2. While the iteration in Algorithm 2 is identical to Garland and Heckbert's (1997) mesh simplification, their contraction point optimisation is unconstrained and therefore much simpler than our Algorithm 1—it could be compared to solving only case $m = 0$ without checking the constraint $P^T v \geq 0$.

4.1.2 Bounding Convex Decomposition

Now that we can generate bounding meshes and reduce the complexity of arbitrary meshes, we can combine our method with Mamou and Ghorbel's (2009) segmentation into convex polyhedra. With the combination of both algorithms, we can generate bounding sets of convex polyhedra of arbitrary geometric models, which is a powerful representation to implement efficient geometric predicates. Mamou and Ghorbel's convex segmentation searches on the dual graph of triangles in order to contract edges, guided by a mixed concavity and aspect ratio cost function. Aspect ratio is defined as the squared perimeter divided by the area of a given mesh, adjusted by a constant factor to yield one in the case of a disk. The cost function is weighted such that the aspect ratio guides the first few iterations of the algorithm, quickly simplifying the mesh. After that, their segmentation is mostly lead by a concavity measure, which they define as the maximum distance of mesh points projected onto the convex hull of that mesh, measured in surface normal direction. When all approximation is disabled, this segmentation step will generate a bounding decomposition into convex polyhedra, however, convex areas (which are very common) are not simplified at all



(a) Original CAD Model

(b) Exact Convex Decomposition

(c) Bounding Mesh

(d) Bounding Convex Decomposition

(e) Overlay on Original Model

Figure 2: Novel bounding mesh and convex decomposition algorithms can handle a typical, non-convex robot mesh with 10^6 vertices (a). Exact decomposition would generate overly high numbers of polyhedra (b), which are inefficient. Our new bounding mesh approximation reduces the number of vertices to 10^3 with an error $\varepsilon < 0.03\,\text{m}$ (c). Then, convex decomposition can simplify this mesh to ≈20 convex polyhedra with 50 vertices each (d, e).

and may contain arbitrary numbers of vertices. As a combination of both algorithms, we first simplify a given model to a bounding mesh for preprocessing (see Figure 2c). After that, we perform the above convex segmentation with all approximations disabled. Finally, we compute the bounding meshes of the convex polyhedra, drastically reducing the number of vertices, and obtain a bounding convex decomposition, as shown in Figure 2e.

Figure 2 shows the compression quality of our bounding convex decomposition through an example: typical robot manipulators can be approximated by 10 to 30 convex polyhedra, totalling no more than 1,000 vertices (Figure 2d). With this, the bounding swept volume of a typical motion of such a robot will simplify to no more than 30–200 convex polyhedra. For instance, in the Bartender scenario, the robot CAD models (86,746 vertices) are first simplified to bounding meshes of 9,030 vertices, and then decomposed into only 22 convex bodies with a total of 963 vertices, all at a bounded geometric error $\varepsilon < 0.03$ m. In contrast to earlier work, we can generate these geometric simplification at a bounded geometric error ε and purely with one-sided (conservative) approximations.

4.2 Bounding Swept Volume Generation

Now that we can represent a static robot's volume by a small bounding set of convex polyhedra, we can also generalise this concept to efficiently compute a bounding swept volume of a robot motion, applying our previous work on non-bounding swept volume generation from (Gaschler, Petrick, Kröger, Khatib, & Knoll, 2013b). In the static case,

(a) A bounding set of convex polyhedra tightly encloses a swept volume of a robot motion.

(b) Overlay on start and end poses of the path.

Figure 3: The swept volume computation can approximate typical robot motions with less than 50 convex polyhedra that enclose the exact swept motion with an error $\varepsilon < 0.05\,\mathrm{m}$. This representation as a set of convex polyhedra allows very efficient geometric queries, using only one-sided (conservative) approximations.

the volume of a rigid body B of a kinematic chain is given by the union of n convex hulls of vertices $\cup_n \mathrm{conv}(V_n)$. In our generalisation, we approximate the swept volume SV of a rigid body $B \in \mathcal{B}$ along a configuration space path Q. In the following, we would like to show the quadratic convergence of our swept volume approximation for motions of serial revolute joints. For that, we first consider the motion of a single convex polyhedron, which is given by a set of vertices V. Let $r_{\max}$ be the maximum distance of a vertex to the first axis, summing up all link lengths and the distance to the last axis. For a single link of length r, a rotation by an angle $\Delta q < \pi/2$ will let a point deviate from the chord (the straight line from start to end) by $\varepsilon = r(1 - \cos(\Delta q/2))$ (Xavier, 2002). For general serial kinematics of multiple revolute joints, Baginski (1997) describes an upper bound for the deviation of the path of a point from the chord of that path:

$$\varepsilon \leq r_{\max} \left(1 - \cos\left(\sum\nolimits_i |\Delta q_i| \;/\; 2\right)\right). \tag{2}$$

Intuitively, this bound is tight when all link lengths but the last one tend to zero and all joints rotate in the same direction around the same axis. (In our implementation, we calculate the bound for each link individually and obtain tighter bounds especially for the first few joints.)

Using the second order series expansion $\cos(\Delta q) \geq 1 - \left(\Delta q^2/2\right)$, we can approximate Eq. 2 as a quadratic function of the angular step size Δq for our swept volume approximation with

$$\varepsilon \leq r_{\max} \left(\sum\nolimits_i |\Delta q_i|\right)^2 / 8. \tag{3}$$

This shows that we may choose an angular discretisation step $\Delta q = \sqrt{8\varepsilon / r_{\max}}$ to generate swept volumes at a desired precision ε, and at a quadratic convergence rate. Contrary to earlier works such as Schulman et al's (2013), we do not suppose a boundary for the screw

motion in each path segment, but rather rely on a more general property of revolute joints given by Baginski (1997).

For swept volume generation, we first discretise the path Q as $q(i)$ at angular steps Δq. Then, as described by Xavier (2002), we generate the swept volume from subsequent convex hulls: for each sweeping convex polyhedron V_n, we compute its convex hull at subsequent path segments $q(i)$ and $q(i+1)$, applying the forward kinematics function FK:

$$\mathrm{SV}(B, Q) = \bigcup_n \bigcup_i \mathrm{conv}\left(\mathrm{FK}(q(i+1))\, V_n \cup \mathrm{FK}(q(i))\, V_n\right). \tag{4}$$

Denoting $|Q|$ as the length of the path in joint space, it follows that we need $\mathcal{O}(n\ |Q|/\sqrt{\varepsilon})$ convex polyhedra to represent the swept volume of a robot motion at a precision of ε, with each one having at most $2\,|V|$ vertices. This result implies that doubling the number of sampling points of the path Q will quadruple the precision of the approximation. Up to now, SV is an approximation with an error bounded by ε. In order to generate a bounding (superset) swept volume and allow our bounded geometric predicates, we can compute ε-enlarged models of B offline and—provided that B is a bounding mesh of the real robot geometry—all computed swept volumes are bounding volumes. This property ensures that our geometric predicates use one-sided, ε-precise approximations as defined at the beginning of this section. As a practical example, the collision query on a robot motion with an object will never return a false negative, and will always confirm a free path when the geometric distance between both is larger than ε. Equivalently, we prepare models of B that are trimmed by length ε to allow inclusion (subset) predicates that never report false positives. Figure 3 shows an example of a typical motion of the Bartender robot, which is simplified to a bounding set of 44 convex polyhedra with an error within $\varepsilon < 0.05\,\mathrm{m}$.

4.3 Bounded Geometric Predicate Evaluation

Considering the definition of our bounded geometric predicates at the beginning of this section, we can now evaluate all collision and inclusion queries using only convex-convex checks, and at a chosen precision. Since all robot and object models in $\mathcal{B}$ are simplified to bounding sets of convex polyhedra, we can rely on efficient algorithms optimised for pure convex-convex queries for both collision and inclusion predicates. An efficient collision checking (and distance query) algorithm for this problem is described by Gilbert, Johnson, and Keerthi (GJK, 1988), which has been shown to detect collisions between two convex polyhedra at a computational complexity linear in the total number of involved vertices. For an environment M of m convex polyhedra, a collision check with a swept robot volume can be performed in at most $\mathcal{O}(nm\,|Q|/\sqrt{\varepsilon})$ time. However, for most practical instances, the computation is leveraged by fast broad-phase algorithms and may be considerably quicker. For example, the geometric world M in the Bartender scenario is rather detailed with a total of 179,092 vertices. The algorithms described in this section simplify this world to 30 convex bodies with a total of ≈600 vertices with an error $\varepsilon < 0.05$ m; in this world, we can evaluate a swept volume collision check in less than 0.01 milliseconds on our testing machine (see Section 7 below). Note that the swept volume collision predicate is by far the most expensive; all other geometric predicates can be evaluated in less than a microsecond.

In addition to the collision and inclusion predicates, there are two predicates available that depend on the forward kinematic function FK of a robot: whether a robot r is at a certain location x, $\mathrm{FK}(r, q) = x$, and whether a robot can reach that location at all, $x \in \Im(\mathrm{FK}(r))$. The first query is implemented by evaluation of the forward kinematic function, the second query relies on an algebraic inverse kinematics solution. For both functions, we use the implementation in the Robotics Library (RL)[2] by Rickert (2011).

5. Knowledge-Level Planning with PKS

High-level planning capabilities in KABouM are provided by the Planning with Knowledge and Sensing system (PKS, Petrick & Bacchus, 2002, 2004), an existing general-purpose planner which builds plans in the presence of incomplete information and sensing actions. PKS is an example of an epistemic planner (Bolander, 2017) which works at the *knowledge level* (Newell, 1982) by reasoning about how the planner's knowledge state changes due to action. In particular, knowledge is described symbolically in a logic-like language and actions are defined in terms of the changes they make to the planner's knowledge state. PKS attempts to avoid the computational explosion that can arise from such methods by limiting the types of knowledge that can be represented, and working directly with the syntactic formulae that model the planner's knowledge, which differs from planners that use representations that enumerate sets of possible worlds or belief states to capture the incompleteness of the planner's knowledge (e.g., Hoffmann & Brafman, 2005). Even though PKS trades expressiveness for tractability, it still supports a rich set of features that are useful in robot environments, including the ability to work with sensing actions, numerical expressions, and uncertain information.

5.1 Knowledge Representation and Reasoning

PKS is based on a generalisation of STRIPS (Fikes & Nilsson, 1971), which describes planning domains using a database mechanism as the underlying state model. Unlike STRIPS, which describes the evolving world state using a single database, PKS uses a collection of five databases, each of which models a different type of knowledge. These databases also have a fixed interpretation in a first-order modal logic of knowledge that formally defines the planner's underlying knowledge state (Petrick, 2006). To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent in each of its databases.

In this work, we primarily use the following three PKS databases:

K_f : This database is like a standard STRIPS database that stores the values of regular fluents that are known to the planner. K_f is primarily used for modelling the effects of actions that change the world. Unlike ordinary STRIPS, PKS does not make a closed world assumption (Reiter, 1978) and K_f instead uses an open world model that explicitly represents both positive and negative facts about the world, and captures the idea that a fact could be unknown to the planner. In particular, K_f can include any ground literal (i.e., a ground predicate or its negation) or ground function (in)equality mapping ℓ, where $\ell \in K_f$ means "the planner knows ℓ."

2. `http://roboticslibrary.org/`

K_w : This database stores information about the effects of sensing actions that return one of two possible outcomes, providing support for modelling information-gathering actions that observe the world but do not necessarily change it. A formula $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$." This disjunction will not be resolved until run time when the action is actually executed in the world, and the planner will receive definite information about ϕ.

K_v : This database stores information about function values that will become known at execution time. In particular, K_v can model the effects of sensing actions that return one of many possible values. K_v can contain any function term f, where $f \in K_v$ means the planner "knows the value of f." As with K_w, the definite value of f will not become known until after the action is actually executed.

PKS also includes two additional databases for modelling a restricted type of disjunctive knowledge (K_x), and localised form of closed world information (LCW) (Etzioni et al., 1994), which are not used in this paper.

Reasoning within PKS is restricted to a set of primitive queries that ask simple questions about the planner's knowledge state. If ϕ is a ground atomic formula and t is a variable-free term then the following primitive queries are permitted: (1) $K(\phi)$: is ϕ known to be true? (2) $K(\neg\phi)$: is ϕ known to be false? (3) $K_w(\phi)$: is ϕ known to be true or known to be false? (Does PKS *know whether* ϕ?) (4) $K_v(t)$: is the value of t known? (The negation of these queries is also allowed.) Primitive queries are evaluated using an inference procedure that checks the contents of the databases and the relationship between knowledge in different databases. Inference in PKS has been shown to be sound, but incomplete (i.e., there are situations where the implemented inference procedure is unable to determine the truth of a query, while an answer would be theoretically possible). Additional details of the reasoning procedure are described by Petrick and Bacchus (2002, 2004, 2006).

5.2 Planning Problems and Plan Generation

A *PKS planning problem* Σ is a tuple $(\mathcal{S}, \mathcal{A}, I, G)$, where $\mathcal{S}$ is a set of *symbols* defining the vocabulary for the planning domain, $\mathcal{A}$ is a set of *actions*, I is a description of the planner's *initial knowledge state*, and G is a set of *goal conditions*. PKS attempts to build a *plan* (a sequence or tree of actions) whose execution in the initial state I brings about a state (or set of states) in which the goal conditions G are satisfied.

$\mathcal{S}$ is a specification of the available objects, predicates, and functions used to define actions and states in PKS. Actions $\mathcal{A}$ are described in terms of their knowledge prerequisites and the changes they make to the planner's knowledge state. Each action $\alpha \in \mathcal{A}$ is a 3-tuple of the form $(\vec{x}, \mathit{pre}(\alpha), \mathit{eff}(\alpha))$. $\vec{x}$ are α's *parameters*, a set of variables that are bound to produce an action instance. $\mathit{pre}(\alpha)$ are α's preconditions, typically described by a conjunctive set of primitive queries about the planner's knowledge state, each of which must evaluate as true before an action can be applied. More complex combinations of primitive queries, including quantified queries and numerical expressions, can also be included in an action's preconditions. $\mathit{eff}(\alpha)$ are α's effects, describing updates to PKS's databases. In particular, effects are specified as a list of statements of the form $\mathsf{add}(\mathcal{D}, \phi)$ (add ϕ to database $\mathcal{D}$) or $\mathsf{del}(\mathcal{D}, \phi)$ (delete ϕ from database $\mathcal{D}$). Conditional effects are also permitted.

Finally, the initial knowledge state I is a specification of the initial contents of each PKS database, and the goal G is an expression consisting of primitive queries, which may include quantified queries.

Given a planning problem Σ, PKS attempts to build a plan to satisfy the goal G by reasoning about the actions $\mathcal{A}$ in a forward-chaining manner. Starting in the initial knowledge state I, if $pre(\alpha)$ of an action α are satisfied by the planner's knowledge state, then $\mathit{eff}(\alpha)$ are applied to produce a new knowledge state. Alternatively, information from PKS's K_w and K_v databases (resulting from sensing actions) can also be used to extend an existing plan, by reasoning about the possible outcomes of such information during plan generation. If a formula ϕ is in the K_w database, denoting the fact that ϕ or $\neg\phi$ will become known at run time, then a *contingent* plan can be built by introducing a pair of conditional branches into the existing plan: along one branch, the planner assumes that ϕ is true, while along the other branch $\neg\phi$ is assumed to be true. (The construction of contingent plans using K_v is similar.) Planning continues until G is satisfied along every branch.

5.3 Externally-Linked Reasoning

To support PKS's internal reasoning processes, the planner also has a mechanism for invoking externally-defined procedures (e.g., special-purpose libraries) during plan generation (Petrick & Gaschler, 2014). This idea is similar to other semantic attachment-like approaches (Eiter et al., 2006; Dornhege et al., 2009a; Erdem et al., 2011), and provides a purely programmatic solution to integrating external reasoning with the planner.

PKS's external evaluation mechanism has the form:

$$\mathsf{extern}(proc(\vec{x})),$$

where *proc* is the name of an external procedure and $\vec{x}$ is the set of parameters that are passed to *proc*. Syntactically, *proc* is a unique identifier to a particular externally-defined procedure; the procedure itself is implemented apart from PKS, for instance in a separate library. A software-level interface function extending PKS's application programming interface (API) (Petrick, 2015) allows the software developer to link *proc* to the actual externally-defined procedure. Intuitively, when an extern call is encountered by PKS, control is transferred to *proc* by executing the externally-defined procedure linked to this identifier, with control returning to PKS when the procedure has terminated. A variant of this mechanism,

$$\mathsf{extern}^{*}(proc(\vec{x})),$$

provides some control over the nature of the external reasoning process, by directing PKS to save the results of the call for the given input parameters $\vec{x}$. Subsequent calls to extern* using the same set of parameters are therefore guaranteed to produce the same output, and the planner is directed to simply used the previous result, which is cached. Such a facility is particularly useful in cases where the external process may be computationally expensive.

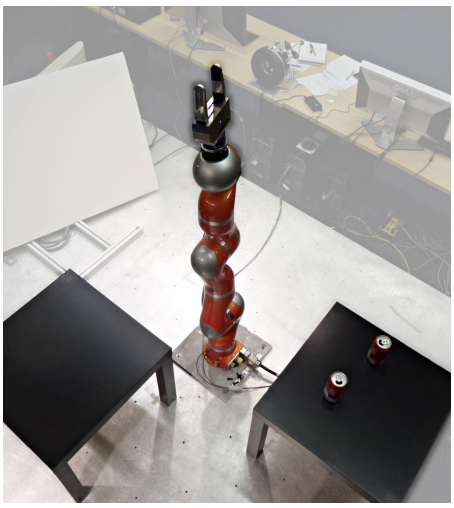
An extern call can appear as part of a primitive query or a database update, where the result of the extern call, defined within the external procedure, is passed back to PKS and interpreted in the context where it occurs. In general, references to symbols in PKS's knowledge state can be directly passed to the external procedure through the argument list $\vec{x}$ of *proc*, providing a connection between the planner's internal knowledge state and

action senseWeight(?o : object)
  preconds:
    K(isGrasped(?o))
  effects:
    add(K_w, isSpillable(?o))

action transferUpright(?o : object)
  preconds:
    K(isGrasped(?o)) &
    K(¬isRemoved(?o)) &
    K(isSpillable(?o))
  effects:
    add(K_f, isRemoved(?o))

action grasp(?o : object)
  preconds:
    K(emptyGripper) &
    K(¬isRemoved(?o))
  effects:
    add(K_f, isGrasped(?o)),
    add(K_f, ¬emptyGripper)

(a) Example PKS actions.



(b) Implementation with a 7-DoF torque-sensing manipulator and a force-controlled parallel gripper.

Figure 4: In the Force Sensing scenario, a compliant robot manipulator can sense if beverage containers are filled by weighing them. To prevent spilling, the robot can hold a container upright while moving, unless it is known to be completely empty or not opened (Gaschler et al., 2013c).

the external reasoning procedure. The execution of extern calls can also be conditioned by including primitive queries as guards, so that external procedures can be context dependent.

In practice, extern calls are most beneficial when used for complex reasoning that cannot easily be modelled in PKS's representation language, or where more efficient reasoning engines already exist. However, unlike PKS's primitive query reasoning mechanism, the planner does not have an internal model of the externally-defined procedures. As a result, it simply assumes that the outcome of an external procedure is sound. Similarly, there is no guarantee that the extern mechanism is complete, since completeness depends on the implementation of individual procedures outside the context of the planner itself. Moreover, in the case of poorly implemented external modules, or difficult problem instances arising from certain input parameters, termination itself may not be guaranteed (or at least guaranteed within a reasonable amount of time). Thus, the onus is strongly placed on the programmer of external procedures to ensure their operation is correct in the planning context.

5.4 Definitions of the Example Domains

Using PKS's representation language, we can now describe how the example domains from Section 3.2 are modelled. Both the Force Sensing and Bartender scenarios are pick-and-place domains where a number of bottles are to be removed from a table, but rely

```
action pickUp(?r : robot, ?o : object, ?l : location)
  preconds:
    K(?l = getObjectLocation(?o)) &
    K(handEmpty(?r)) &
    K(extern(isReachable(?l, ?r))) &
    forallK(?other:object)
      ¬K(extern(graspMotionCollides(?l, ?r, ?other)))
  effects:
    del(K_f, ?l = getObjectLocation(?o)),
    del(K_f, handEmpty(?r)),
    add(K_f, inHand(?o, ?r))

goal forallK(?o:object) (
      K(extern(includes(dishwasher, getObjectLocation(?o)))) |
      K(¬isEmptyBottle(?o)) )
```

Figure 5: Example PKS action and goal for the Bartender scenario.

on different types of sensing, force sensing and visual sensing.

Figure 4 shows three PKS actions in the Force Sensing scenario, along with an image of the operating environment. The first action, senseWeight, is an example of a sensing action that checks the weight of an object ?o to determine whether it is spillable or not. As a precondition for applying this action, the object must first be grasped by the robot (i.e., isGrasped(?o) must be known to the planner). After execution, the planner comes to know whether the object is spillable or not (i.e., isSpillable(?o) is added to the K_w database)—information which the planner can use to build contingent branches into a plan and reason individually about the two possible outcomes of the sensing action.

The second action, transferUpright, is a manipulation action which removes a spillable object ?o from a surface. In order to apply this action, the robot must first be holding the object, the object must not have been previously removed, and it must be spillable (i.e., isGrasped(?o), ¬isRemoved(?o), and isSpillable(?o) must be known). As an effect, the action removes the object from the surface (i.e., isRemoved(?o) is added to the K_f database), following a trajectory which keeps the object upright.

Figure 5 shows a manipulation action, pickUp, from the Bartender scenario, which uses a specific robot hand ?r to pick up an object ?o from a location ?l. This action is modelled by a set of preconditions that verify that the planner knows the location of the object, that the robot hand is empty, that the location is reachable with the robot hand, and that the swept volume of the motion does not collide with any other objects. In particular, determining that a location is reachable, and that no collisions will occur, involves invoking the external path planner, represented in the action description by two extern calls (to the procedures isReachable and graspMotionCollides, respectively) which are evaluated immediately at planning time. For instance, K(extern(isReachable(?l, ?r))) represents an external call that invokes the path planner to determine whether location ?l is reachable using robot hand ?r; this call would evaluate as true provided the path planner evaluates isReachable(?l, ?r) as true. After applying this action, the planner comes to know that the object is no longer at its former location, the robot hand is no longer empty, and that the

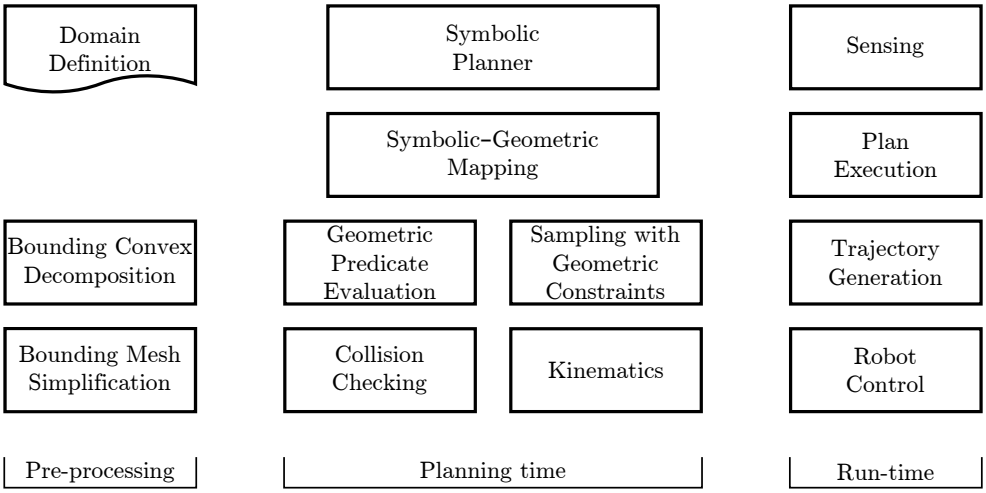


Figure 6: Overview of the implemented KABouM software architecture.

robot is holding the object.

Finally, Figure 5 also gives an example of a goal in the Bartender scenario. In this case, PKS is tasked with constructing a plan to ensure that all empty bottles are located in the "dishwasher" area. In particular, reasoning about whether an object is in the dishwasher or not is transferred to an externally-defined procedure, again represented in PKS by an extern call (to the procedure includes).

The above action and goal definitions will be used below in Section 7 to evaluate our framework. We note that the domain descriptions we present here are not complete,[3] but demonstrate important features of our approach, such as how sensing actions are used, and how geometric functions for collision checking and motion planning are integrated into our scenarios at the planning level. We also note that since PKS is a general-purpose planner which has not been specifically optimised for robot domains, these domain descriptions may easily be generalised to other robot tasks, since they involve common object transfers and collision-avoiding robot motions.

6. KABouM System Architecture

The components described above—the symbolic planner, the domain definition, and the geometric volume computation and predicates—account for only part of the KABouM framework. In order to test the effectiveness of our approach, we have implemented and evaluated our framework both in simulation and on several real robot systems. Figure 6 shows an overview of the implemented software system architecture.

From an abstract perspective, the underlying design goals of KABouM involve: (i) keeping the domain definition and domain-specific software separate from generic task planning, (ii) providing a consistent framework from task definition to physical execution, and (iii)

3. The Force Sensing scenario also includes an action transfer for moving objects that aren't spillable, as well as the necessary ungrasp action that follows all transfer actions. The full definition of the Bartender scenario also includes an action senseIfEmpty which senses whether a bottle is empty or not, and an action putDown which allows the robot to place an object at a specific location using a particular robot hand.

allowing efficient evaluation of geometric and kinematic predicates. For these reasons, we did not rely on a large-scale robotics framework such as the Robot Operating System, but rather cherry-picked individual kinematics and geometry functions from existing libraries, and implemented a consistent task planning framework.

As shown in Figure 6, the system follows a sequence from the problem-dependent domain definition and pre-processing of the geometry to generic task and motion planning and execution on physical robots. During planning, the PKS planner can generate motion plans and check for collisions by calling external predicates from the motion planning and collision detection components. Both components rely heavily on the Robotics Library (RL) by Rickert (2011). RL includes wrappers for all state-of-the-art collision checking libraries and a manually optimised algebraic solution of the inverse kinematics for all six degrees-of-freedom robots that appear in our evaluation.

For the simpler scenarios, robot motion paths are not searched completely, but generated by heuristics. In the Remove n Objects scenario, paths are joint-space interpolations between start and goal. In the Force Sensing, paths are interpolated in the operational space to achieve the transferUpright action. In the Bartender scenario, paths are constructed from a set of collision-free waypoints. Even though path generation is practical in these scenarios, it is not guaranteed to search all possible paths.

By contrast, the Bimanual Assembly scenario is solved with fully integrated task and motion planning. To integrate symbolic and geometric planning, the symbolic planner needs to understand which geometric states are reachable through which actions, but does not need to understand the meaning of those states. For this, a bidirectional map between geometric states and corresponding symbols in the planner is sufficient. Inspired by Dornhege (2014), our symbolic state contains a tuple of symbols $\mathcal{S}_1 \times \mathcal{S}_2 \times \ldots$ that uniquely maps to the geometric state. In the current implementation, each entry of this tuple identifies a robot or an object configuration. A unique map is achieved by using a lexicographic order on the list of known geometric states. When the symbolic planner progresses the search and has found an action whose symbolic preconditions are fulfilled, it calls a robotics-specific function with (parts of) its current geometric state symbols (and possibly other symbolic arguments) as parameters. In our implementation, PKS calls robotics-specific functions through its extern interface, and can pass parameters that map to parts of the geometric state. The robotic-specific function then progresses the geometric states, evaluates for geometric predicates, and returns a tuple of symbols for the new geometric state. Translation between geometric states and their symbols are performed by a symbolic-geometric mapping component. It only requires that new symbols of a certain type or natural numbers can be saved in the planner's state, and can therefore interact with most off-the-shelf planners.

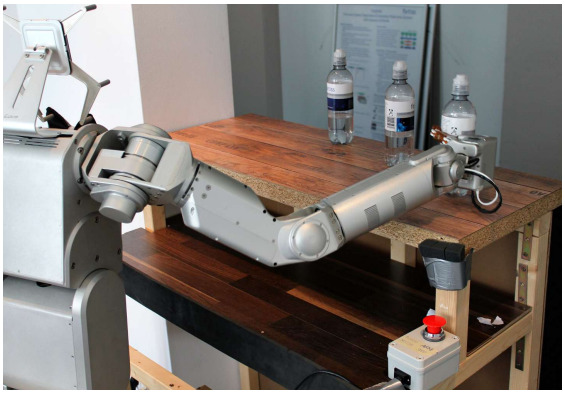
Note that this approach requires an unconditional action that samples new robot and object states at random for covering the search space, so that the planner can sample additional geometric choices for the same symbolic action. To accomplish this, we define a move action that samples random transit and transfer motion. Actions for picking and placing also need to cover their respective subspaces, which we achieve by sample-project algorithms (Gaschler, 2016, ch. 5).

At run time, action execution is mediated by a simple plan execution component, which sequentially processes the symbolic plan, given as a tree of actions, starting from the root. Binary sensing actions have a choice of a positive and a negative child, which is selected af-

```
action remove(?o : object)
  preconds:
    K(¬isRemoved(?o)) &
    forallK (?p : object)
    (K(isRemoved(?p)) |
    ¬K(extern*(graspMotionCollides(?o, ?p))))
  effects:
    add(K_f, isRemoved(?o))

goal: forallK (?o : object)
    (K(isRemoved(?o)))
```

(a) Symbolic domain definition.



(b) Implementation with a Meka H2 manipulator with $n = 3$.

Figure 7: Remove n Objects scenario.

ter execution; all other actions have a single child (Petrick & Bacchus, 2002, 2004; Gaschler et al., 2013c). Sensing components are specifically implemented for each scenario: the Bartender scenario includes a simple colour segmentation for object recognition, while the Force Sensing robot reports external forces by computing its inverse dynamic model. While the current implementation simply executes the plan, future versions may be integrated with the execution monitoring features of PKS (Petrick & Foster, 2013). For executing robot actions, the trajectory generation and robot control components again rely on the RL library to interpolate paths by quintic polynomials. Finally, robot control directly commands robot joint angles through hardware-specific drivers.

6.1 Solution of the Remove n Objects Scenario

In order to help understand the operation of the KABouM architecture, we discuss another minimalistic scenario step-by-step, the Remove n Objects scenario, in order to demonstrate a typical interaction of symbolic planning and geometric predicates. In the Remove n objects scenario, a single manipulator must remove a number of objects from a table while avoiding collisions (see Figure 7b). Typically, many grasp poses are obstructed by neighbouring objects but, at any time, at least one object can be removed safely. Essentially, the task planner has to find a sequence of remove actions where collisions between objects can be avoided, as defined in Figure 7a.

In the KABouM architecture, almost all components are kept generic and separate from problem-specific definitions. In the Remove n objects scenario, the domain definition files contain the symbolic domain (Figure 7a), the kinematics and geometric models of the robot, the geometry of the objects and the environment, and a grasping pose for the robot hand and the particular type of object. Only the robot control component was adapted to this type of robot, a very thin external predicate graspMotionCollides was implemented that uses several geometric predicates described in Section 4, and the initial positions of the objects and a list non-colliding target "storage" positions were provided to the planner.

Before actual task planning, KABouM generates bounded convex decompositions of all

geometric models. Then, PKS performs a forward search on the symbolic domain; in this scenario, it evaluates the preconditions of the remove action using different objects ?o as an argument. Besides checking the purely symbolic precondition that a previously removed object may not be removed again, there is a hybrid predicate over all other objects ?p that they must not collide with the motion of removing object ?o. As an effect, the remove action sets the symbolic predicate isRemoved as known for that object, getting closer to the goal criterion that all known objects are known to be removed. In order to evaluate the external predicate graspMotionCollides, PKS makes a direct call to a simple domain-specific function which computes the grasping pose to the given object and to a target location using KABouM's inverse kinematics predicate, and tests whether the swept volume from the grasping pose to the target pose would collide with another object ?p, using KABouM's swept volume collision predicate defined in Section 4. The extern call specifies that the return value of this predicate depends purely on its arguments, enabling caching. Besides this, all interpolated paths, swept volumes, and collision query results are also cached.

At run-time, the plan execution component of KABouM iterates through the solved plan, retrieves the cached paths of the actions, generates trajectories through a simple quintic interpolation scheme, and finally executes these trajectories on the physical robot.

7. Evaluation

In order to evaluate our approach, we discuss typical solutions of the Bartender and Force Sensing scenarios, which illustrate many of the features of KABouM. In addition, we analyse the performance and scalability of our approach on two scenarios that allow arbitrary numbers of objects, namely the Remove n Objects and the Stacked n Objects scenarios. Finally, we discuss Bimanual Assembly scenario, which is solved with fully integrated task and motion planning.[4]

All plans were generated on desktop computers with a dual-core 2.8 GHz processor and 8 GB of memory. While the planning software was similar in all scenarios, each scenario included a different type of physical robot. Because of the modular KABouM architecture, we could encapsulate the different robot and sensing hardware interfaces within the robot control and sensing components, respectively. On the planning level, these diverse types of robots also indicate that KABouM is not limited to a single type of task, but can also solve rather generic tasks, including mobile and multiple-manipulator robots.

7.1 Bartender Scenario

The Bartender scenario was implemented on a two 6-DoF industrial manipulator setup with a colour camera for object recognition. The problem instance evaluated involved a setting with four bottles on a table (Figure 10), whose positions were automatically measured by a visual background-subtracting object recognition system before actual task planning. Offline geometric mesh simplification and convex decomposition took 27.4 seconds, and task planning took 2.9 seconds (Gaschler et al., 2013a). Figure 10 shows the executed sequence of actions, which corresponds to a path in the solved plan: first, image recognition is directed

4. The domain definitions of the evaluated scenarios will be made available on the first author's website `http://www.andre-gaschler.de/` at the time of publication.

Table 1: Evaluation of the Remove n Objects scenario. Geometric queries such as collisions are made efficient by the swept volume representation as sets of convex bodies. The overall planning time is on the order of typical execution time and scales acceptably with the number of objects in this scenario.

Number of Objects n	Total Time [s]	Inverse Kinematics [s]	Path Planning [s]	Swept Volume Generation [s]	Number of Triangles Generated	Number of Convex Bodies Generated	Number of Collision Checking Calls	Collision Checking [s]	Number of Geometric Precondition Queries	Number of Geometric Effect Evaluations	Number of Symbolic Actions in Solution
Objects at random locations											
2	12.41	0.01	12.21	0.19	11500	401	1	.00001	3	2	2
3	40.35	0.01	39.72	0.62	36956	1290	4	.00034	6	3	3
4	39.48	0.01	38.74	0.70	43162	1490	15	.00111	10	4	4
5	102.30	0.03	101.46	0.81	47570	1657	20	.00146	15	5	5
10	143.67	0.04	141.74	1.84	110302	3819	276	.01712	120	15	10
15	284.44	0.07	282.14	2.21	131396	4565	443	.02114	210	20	15
20	356.98	0.10	354.01	2.81	169396	5885	764	.04442	320	64	20
Objects in a line, only 1 of n can be picked up											
2	12.43	0.01	12.23	0.19	11500	401	1	.00001	3	2	2
3	39.87	0.01	39.44	0.42	24768	860	3	.00003	6	3	3
4	99.46	0.02	99.01	0.43	24768	860	6	.00004	10	4	4
5	158.70	0.02	158.26	0.42	24768	860	10	.00005	15	5	5
10	351.77	0.02	350.67	1.08	61676	2156	79	.00270	55	10	10
15	354.10	0.05	352.13	1.89	113290	3927	322	.01275	120	15	15
20	355.16	0.12	352.43	2.57	153936	5331	792	.02974	210	20	20

to determine which bottles are empty, which involves four sensing actions. After following the appropriate branches in the contingent plan generated by the planner, which considers the possible outcomes of the sensing actions, the rest of the plan is sequential and consists of six pick-and-place actions. Interestingly, this simple bimanual robot scenario gives rise to non-trivial behaviour: in order to move bottles from the right side of the bar to the goal location, one arm must first move them to a location where the second arm can reach them, as shown in Figure 10. This behaviour is not hard-coded a priori but is generated as a result of the planning process.

7.2 Force Sensing Scenario

The Force Sensing domain is very similar to the Bartender domain, with the visual recognition of empty bottles replaced by internal torque sensing of a single, compliant robot manipulator. This is an example of interdependent sensing and manipulation actions, as the manipulator needs to grasp the object in order to weigh it (Gaschler et al., 2013c). In contrast to the Bartender scenario, which includes a visual sensing action, force sensing

grasp(can1)
senseWeight(can1)
branch(isSpillable(can1))
K+:
  transferUpright(can1)
  ungrasp(can1)
  grasp(can2)
  senseWeight(can2)
  branch(isSpillable(can2))
  K+:
    transferUpright(can2)
    ungrasp(can2)
  K-:
    transfer(can2)
    ungrasp(can2)
K-:
  transfer(can1)
  ungrasp(can1)
  (truncated for brevity)

(a) Example solution

grasp(can1)
senseWeight(can1)
branch(isSpillable(can1))

transferUpright(can1)

ungrasp(can1)

grasp(can2)
senseWeight(can2)
branch(¬isSpillable(can2))

transfer(can2)

ungrasp(can2)

(b) Action and branch resolution sequence

Figure 8: Force Sensing scenario: example solution for two objects. Depending on the weight measurements of the grasped objects at run time (images **A** and **D**), different branches in the generated plan are taken. Only when a container is known to be empty, can it be transferred using a fast, arbitrary path (image **E**). A video of this scenario is available at `http://youtu.be/7l2NP3l9_lY`.

can only be applied to objects that are grasped. Therefore, sensing actions and binary branches are moved from the start of the plan to several actions later, when the isGrasped precondition is fulfilled. Figure 8 shows an example contingent plan for two objects to be transferred. The KABouM execution component follows the actions in the solved plan and chooses appropriate branches by assessing the results of sensed information. This scenario was physically implemented on a joint-impedance controlled light-weight 7-DoF robot with a force-controlled parallel gripper (Figure 8b). The positions of objects were not recognised automatically, but included into the domain definition. The external gravity force of objects was measured by internal torque sensing and calculating inverse dynamics.

While the Force Sensing and Bartender scenarios serve as examples that highlight various features of KABouM, we also analyse the scalability of our approach below. To do this, we quantitatively evaluate two scenarios that can easily be defined with an arbitrary number of objects: the Remove n Objects and the Stacked n Objects scenarios. While both scenarios are pick-and-place problems with few types of actions, their underlying search spaces are substantially different, providing insights into which problem types scale well in KABouM, and which are potentially problematic.

Table 2: Evaluation of the Stacked n Objects scenario, where n objects are to be stacked in a certain order. This seemingly simple scenario is a hard problem, as only a global search can find a correct plan, and plans are sparse in the search space because of the ordering constraint.

Objects	Total Time	Symbolic Planning	Inverse Kinematics		Path Planning	
n	[s]	[s]	[s]	Calls	[s]	Calls
2	4.5080	0.0137	0.14031	20	4.3315	18
4	8.4666	0.0730	0.14138	16	8.2029	42
6	15.0111	2.0692	0.14471	24	12.7248	66
8	80.7198	64.1298	0.33280	30	16.8952	91
9	$>$ 300 (timeout)					

7.3 Remove n Objects Scenario

As discussed in the previous section, the task in the Remove n Objects scenario is simple: a single robot should remove objects from a table while avoiding collisions. Typically, many grasp poses are obstructed by neighbouring objects but, at any time, at least one object can be removed safely. Thus, the task planner has to find a sequence of remove actions where collisions between objects can be avoided. We implemented this scenario in a humanoid robot setup with a 2-DoF Meka Robotics H2 torso, a 7-DoF arm and a tendon-driven humanoid hand, as shown in Figure 7. In our demonstration, the robot removes three objects from previously known locations.

Besides this simple demonstration, we perform a more quantitative evaluation in a simulated environment, measuring the performance of the planner with respect to the number of objects n. Contrary to the scenarios studied earlier, problem instances were generated systematically: random locations were evenly distributed over a rectangular table area until a non-colliding set was found, locations in a line were generated with a defined starting position and increment. As shown in the results in Table 1, the planning time is almost linear in the number of objects. In general, planning time is on the same order of magnitude as the typical execution time on the robot. Even though the number of collision tests may be higher than quadratic in the worst case, it must be noted that collisions of convex polyhedra can be checked very efficiently, and collision checking only amounts to a negligible fraction of the total planning time for the numbers observed. Swept volume generation time mostly depends on the number of objects grasped, and only slightly on the location of those grasps. In the extreme case of all objects standing in a straight line, only one of the objects can be picked up at a time. Even though placement of the objects has some effect on path planning and collision testing, influence on the total planning time is minor. As a result, our planning approach scales well for this simple scenario, where only one type of action is allowed that is not reversible, and a geometric predicate needs to be checked only once for each pair of objects.

7.4 Stacked n Objects Scenario

In the Stacked n Objects scenario, a single mobile manipulator must move a new block under an existing stack of $n-1$ blocks, keeping the original order. As shown in Figure 9, a single, grey block is located at the centre location, while all other $n-1$ blocks are stacked at the left location. The robot can move a block from the top of a stack to the top of another, and is supposed to move the blocks such that the grey block is under the original stack in the final configuration. While this simple pick-and-place domain involves only n objects and a fixed number of 3 locations, it is rather hard to solve. The results of this example are shown in Table 2. Since the number of objects and locations is limited, geometric predicates can be evaluated within reasonable time. However, the number of possible symbolic states grows super-polynomial, and states that fulfil part of the goal criterion are many actions away from the single goal state. (When only a single block is wrongly placed at the bottom of the stack, this state is as far from the goal state as the initial state in terms of numbers of actions.) For this reason, the symbolic planner essentially has to search to entire search space, and for $n > 8$, the domain becomes infeasible on our testing system due to memory constraints.

In principle, this scenario is a classical example of Sussman's Anomaly (Sussman, 1973), in that a global search is needed to achieve a correct solution. The underlying challenges are that choices early in the plan may render the goal infeasible, correct plans are very sparse in the search space, and states fulfilling part of the goal criterion are possibly far away from the single goal state. We voluntarily included this example to show current limitations of our implementation, and to motivate future work.

7.5 Bimanual Assembly Scenario

While the previous scenarios highlight individual aspects of our framework, we finally discuss an integrated scenario with two kinematics and multiple types of actions and objects, the Bimanual Assembly scenario. In this scenario, two industrial manipulators are to assemble a gearbox component from four objects (Figure 11). Besides the usual pick-and-place actions, four complex actions for assembly and object handover are available, two of them requiring coordinated dual-arm motion. The definition of predicates and actions are given in Tables 4 and 5 in the Appendix A; implementation details are discussed by Gaschler (2016, ch. 6).

To assemble the gearbox component, four objects of three different types are needed, two bearings, a pipe object, and a mechanical tree (Figure 11 a–c). Two subcomponents

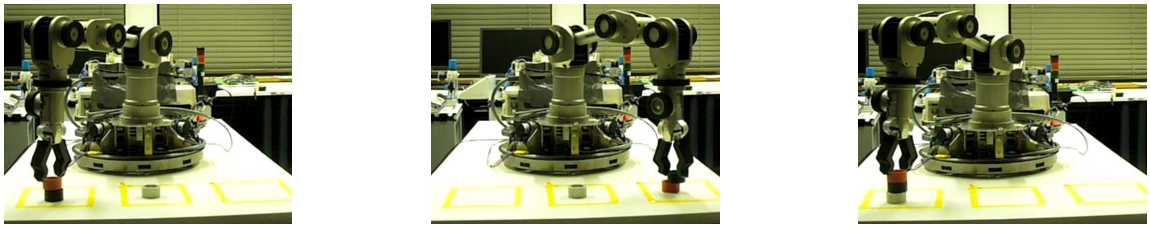


Figure 9: The Stacked n Objects scenario was implemented on a mobile manipulator with $n = 3$.

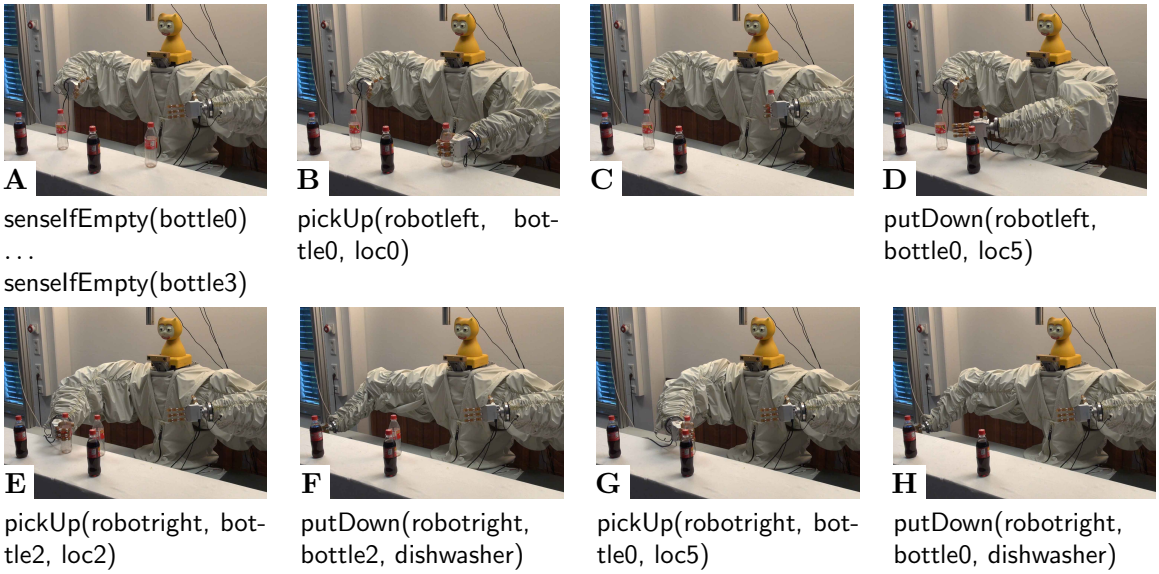


Figure 10: Action sequence of a solution in the Bartender scenario. In this example, the left arm is the only one to reach bottle0 (image **B**). However, only the right arm can reach the desired dishwasher location. Therefore, the robot's left arm moves the first bottle to a location where the other arm can reach it (image **D**), and the right arm passes it on to its final location (images **G** and **H**). We note that this behaviour has not been preprogrammed but instead arises purely from symbolic planning (Gaschler et al., 2013a). A video of this scenario is available at `http://youtu.be/yMmZkhHr8ss`.

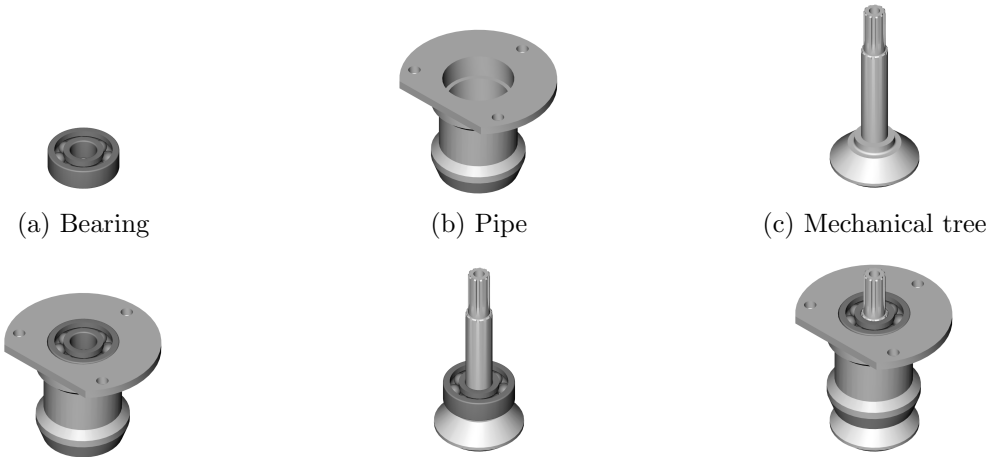
need to be created from two objects each, and these subcomponents can be assembled to form a gearbox. The two robots have different gripper fingers: one can grasp all objects from the outside, the other one can grasp a bearing object from the inside. The latter grasp is necessary to insert a bearing into a pipe, and additionally allows the two robots to hand over a bearing from one to another. Because of low tolerances, some assembly actions require both objects to be grasped. In total, seven different actions are available (Figure 12 c–h), two of which involve bimanual manipulation. While the move action does not have preconditions and can always sample new poses at random, all other actions sample poses that are generated by iterative projection starting from the current pose. With this set of actions and sampling routines, the full space of robot and object configurations is covered, including the rotational degree-of-freedom for grasping and assembling cylindrical objects.

To evaluate this scenario, we compare our bounded geometric predicates with traditional collision checking, vary between three search schemes, and measure computation time of our planning system and its components. The measurement results are listed in Table 3.

As a first observation, all search schemes succeed at solving the Bimanual Assembly scenario with and without using a bounding convex decomposition of the geometry. Bounding geometric predicates allow the planning system to solve this scenario in one tenth of the time compared to conventional collision checking, irrespective of the search scheme. As a result, bounding geometric predicates are more efficient than normal collision checks, using

Table 3: Evaluation of the Bimanual Assembly scenario (Gaschler, 2016). Computation time and procedure calls are measured with respect to the search strategy of the symbolic planner and the type of geometry. Averages are taken over 12 trials, standard deviation is shown in gray.

	Depth-first Search		Breadth-first Search		Iterative Deepening Search	
Bounded geometric predicates $\varepsilon < 0.02\,\mathrm{m}$ on a bounding convex decomposition of the scene geometry						
Total Time [s]	0.737	±0.429	13.532	±0.933	11.730	±0.413
Symbolic Planning [s]	0.009	±0.005	0.154	±0.015	0.130	±0.005
Geometric Search [s]	0.728	±0.424	13.378	±0.918	11.600	±0.408
Inverse Kinematics Calls	191.75	±97.68	3621.50	±359.78	3194.25	±150.89
Inverse Kinematics [s]	0.001	±0.001	0.018	±0.001	0.015	±0.001
Collision Checks	2845	±1600	56770	±4547	49169	±1826
Collision Checking [s]	0.713	±0.416	13.132	±0.893	11.382	±0.396
Positive Collision Checks [%]	10.71	±2.88	6.31	±0.60	6.84	±0.43
Geometric States	48.50	±32.15	322.00	±174.51	133.25	±30.45
Actions in Plan	26.75	±7.04	6.00	±0.00	7.00	±0.00
Waypoints in Plan	53.50	±11.84	21.00	±0.00	22.00	±0.00
Variant: Exact geometric predicates on the original scene geometry						
Total Time [s]	11.898	±7.212	322.049	±23.507	249.637	±20.253
Symbolic Planning [s]	0.008	±0.005	0.277	±0.017	0.209	±0.019
Geometric Search [s]	11.889	±7.207	321.772	±23.494	249.427	±20.235
Inverse Kinematics Calls	140.00	±86.98	3791.00	±258.85	3096.00	±318.01
Inverse Kinematics [s]	0.001	±0.001	0.037	±0.002	0.028	±0.003
Collision Checks	1899	±1248	59293	±4136	47699	±3919
Collision Checking [s]	11.874	±7.199	321.403	±23.475	249.135	±20.205
Positive Collision Checks [%]	8.07	±1.33	6.55	±0.21	7.95	±2.05
Geometric States	35.00	±20.31	362.75	±149.95	108.75	±69.56
Actions in Plan	25.25	±2.50	6.00	±0.00	7.00	±0.00
Waypoints in Plan	48.75	±3.50	21.00	±0.00	22.00	±0.00



(a) Bearing (b) Pipe (c) Mechanical tree

(d) Subassembly of pipe and bearing (e) Subassembly of mechanical tree and bearing (f) Gearbox component assembled from both subassemblies

Figure 11: Objects in the Bimanual Assembly scenario. The final gearbox is assembled from two bearing objects, a pipe object, and a mechanical tree object.

the state-of-the-art collision checking library Bullet.

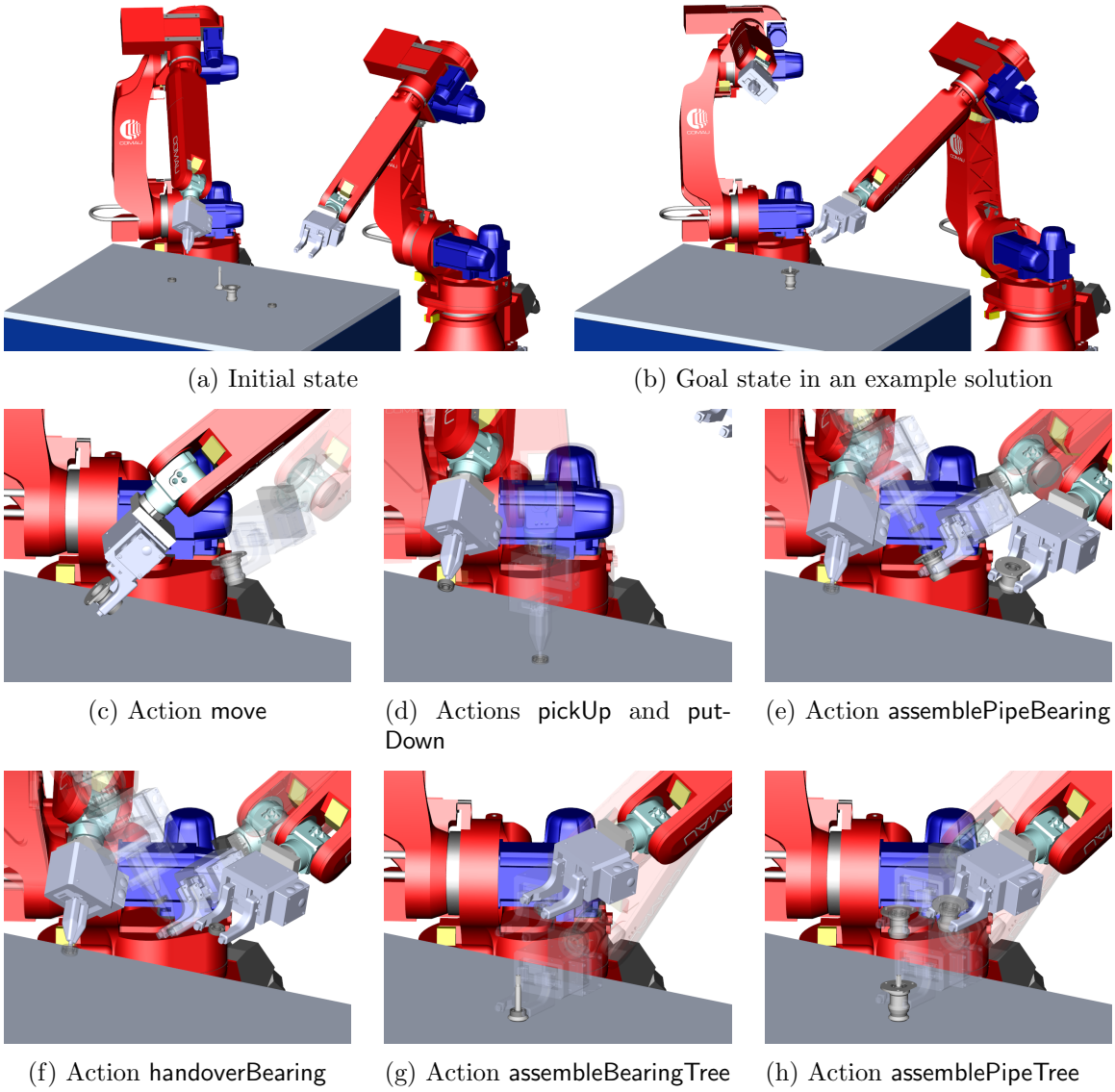
Considering the different search schemes, a depth-first search is fastest in this scenario, which could be explained by the fact it never revisits states before an assembly action. However, depth-first solutions contain many unnecessary pick-and-place actions, while the breadth-first search scheme always finds a plan with the minimum length of six actions.

Overall, the Bimanual Assembly scenario shows how integrated task and motion planning provides a complete search for domains defined on a task level. Solutions include automatic re-grasping, sequences of grasps, and even coordinated bimanual assembly actions. Preconditions such as reachability and collision-free motion allow abstract action definitions, and ultimately ensure the correctness of multi-robot behavior more complex than what can be programmed by hand.

8. Discussion

This work makes two main contributions: the introduction of single-sided ε-precise geometric collision checks, and an alternative approach to integrating task and motion planning with discrete uncertainty.

In contrast to earlier approaches, our collision checks operate on a bounding convex decomposition of the geometry. This single-sided approximation allows better motion planning performance than standard triangle meshes (which are not as fast), convex hulls (which are not as precise and overlook thin passages), and approximate convex decompositions (which are often incorrect and ignore certain collisions). Few approaches deal with single-sided approximation of meshes (Sander et al., 2000; Platis & Theoharis, 2003), and none of these apply to the problem of efficient and strict collision avoidance. By segmenting the geometry into convex bodies and simplifying these with our bounding mesh algorithm, we generate a single-sided ε-precise approximation that allows fast, safe, and accurate collision checks.



(a) Initial state

(b) Goal state in an example solution

(c) Action move

(d) Actions pickUp and putDown

(e) Action assemblePipeBearing

(f) Action handoverBearing

(g) Action assembleBearingTree

(h) Action assemblePipeTree

Figure 12: Examples of initial state, goal state, and actions in the Bimanual Assembly scenario. Transparent colours indicate waypoints before the resulting state of an action.

Compared to the conventional approach of using state-of-the-art collision checking libraries on triangle meshes, we achieved a speed-up of a factor of ten. On the predicate level, collisions between continuously moving robots and objects can be formulated in terms of swept volumes; this representation is inspired by Kaelbling and Lozano-Pérez's work.

In terms of combined task and motion planning, our work adds to the small (but growing) number of robotics-specific approaches that use general-purpose automated planning under (discrete) uncertainty (Kaelbling & Lozano-Pérez, 2013): new domains and actions can be specified in a standardised language, with integrated support for common, robotics-specific functions and predicates. While many of the planning approaches described in

Section 2 apply heuristics tailored towards specific domains (Barry, 2013; Plaku & Hager, 2010; Hauser & Ng-Thow-Hing, 2011; Karlsson et al., 2012), where it is cumbersome to define new types of actions, our planning approach has not been specifically optimised for robotics tasks. In addition, almost all of the planning approaches mentioned in Section 2 apply the closed-world assumption (an exception is Kaelbling & Lozano-Pérez, 2013), where all unknown predicates are assumed to be false. Our approach is instead able to reason about incomplete knowledge, where certain facts may be unknown to the planner, and model situations where actions produce information gain and loss. Furthermore, while our geometric preconditions may be similar to those by Kaelbling and Lozano-Pérez (2011), their underlying hierarchical, back-chaining planning strategy substantially differs from our knowledge-based approach. Its hierarchical search can potentially solve larger scenarios, but relies on the assumption that efficient implementations are given for all preconditions of abstract operations. By contrast, our approach is more modular and only checks preconditions of actions, which are straightforward to implement for a larger range of scenarios.

Considering limitations, KABouM requires full knowledge of the geometric state, while a few planners (Kaelbling & Lozano-Pérez, 2013) can search in the belief space. Compared to more robotics-specific planners (Srivastava et al., 2014), our symbolic planner cannot understand why a collision happened, but can only retry another motion path for the same action or retry another action.

If an action depends only on part of the geometric state, KABouM can use this in symbolic planning, which enables us to solve some problem instances where existing sampling-based planners may run out of time. For instance, it is a general property that sampling-based motion planners fail inconclusively, as the general motion planning problem is PSPACE-complete (Reif, 1979). While sampling-based planners have to apply heuristics in their search (Cambon et al., 2009), KABouM searches on the symbolic level.

It is also important to note that our planning approach does not rely on our particular choice of planner. In addition to PKS's extern mechanism, the use of PKS was also motivated by it's ability to model incomplete information and sensing actions, and the availability of a comprehensive programmer-level API for software integration (Petrick, 2015). While other options do exist for satisfying the final criterion, there are relatively few planners that can reason with knowledge and sensing as PKS does. However, provided another planner provides the same core features as PKS, there is no reason it cannot be used in its place. Indeed, with the release of frameworks like ROSPlan (Cashmore et al., 2015) (which did not exist when we began this work), we have the opportunity to test other domain-independent planners, which we are exploring as future work.

9. Conclusion

In this paper, we describe an approach to robot task planning that combines single-sided approximate reasoning about complex geometric shapes with general-purpose knowledge-level planning techniques. We introduce a set of algorithms for bounded geometric predicates in this framework, and demonstrate our approach using several scenarios involving sensing actions, multiple manipulators, mobile manipulation, and assembly. Overall, we believe that knowledge-level task planning combined with collision checking on a bounding convex decomposition is applicable to a broad range of robot tasks, and may prove effective in struc-

tured and partially known environments, including automation, robot-aided manufacturing, and mobile manipulation, involving arbitrary numbers of manipulators.

We also view the use of a general-purpose, domain independent planner that has not been explicitly optimised for planning robotics tasks as a significant advantage of our approach. First, our framework profits from future updates to the actively-developed PKS planner (or other compatible off-the-shelf automated planners) which may provide improvements to its planning capabilities and new features which could be introduced in KABouM. Second, by treating planning as a black box we keep our framework sufficiently modular, allowing us to consider other symbolic planners from the planning community which may be tested in the KABouM framework with minimal effort. Finally, since we use a general-purpose planner, we are also exploring applications of our reasoning framework to other robot tasks, for instance generating natural language dialogue for human-robot interaction (Petrick & Foster, 2013, 2016).

Acknowledgements

The authors would like to thank Torsten Kröger, Svetlana Nogina, Quirin Fischer, and Sören Jentzsch for their help with the implementation and evaluation, and Markus Rickert for his comments on the manuscript. This research was supported in part by the European Commission's 7th Framework Programme through grant no. 270435 (JAMES: Joint Action for Multimodal Embodied Social Systems, `http://james-project.eu/`).

Appendix A. Definition of the Bimanual Assembly Scenario

Table 4: Symbolic predicates and problem instance definition of the Bimanual Assembly scenario.

Domain Element	Element Definition
Types	object, robot
Constants	object o_1, o_2, o_3, o_4 robot $robot_1$, $robot_2$
Predicates	isGrasped, isRobotGrasps, isHandEmpty, isContained, isABearing, isAPipe, isATree, isAGearbox, containsBearing, isContained
Initial knowledge K_f	isABearing(o_1), isABearing(o_2), isAPipe(o_3), isATree(o_4), isHandEmpty($robot_1$), isHandEmpty($robot_2$), ¬isContained(o_1), ¬isContained(o_2), ¬isContained(o_3), ¬isContained(o_4), ¬isGrasped(o_1), ¬isGrasped(o_2), ¬isGrasped(o_3), ¬isGrasped(o_4)
Goal criteria G	exists(object o) (K(isAGearbox(o)) ∧ K(¬isGrasped(o)))

Table 5: Symbolic action definition $\mathcal{A}$ of the Bimanual Assembly scenario.

Action	Preconditions	Effects
move(robot r)	true	(no symbolic effects)
pickUp(robot r, object o)	K(isHandEmpty(r)) $\wedge$ $K(\neg$isGrasped(o)) $\wedge$ $K(\neg$isContained(o))	add(K_f, $\neg$isHandEmpty(r)), add(K_f, isGrasped(o)), add(K_f, isRobotGrasps(r, o))
putDown(robot r, object o)	K(isRobotGrasps(r, o)) $\wedge$ K(isGrasped(o)) $\wedge$ $K(\neg$isContained(o))	add(K_f, isHandEmpty(r)), add(K_f, $\neg$isGrasped(o)), del(K_f, isRobotGrasps(r, o))
assemble PipeBearing(robot r, object o, robot r', object o')	K(r $\neq$ r') $\wedge$ K(isRobotGrasps(r, o)) $\wedge$ K(isRobotGrasps(r', o')) $\wedge$ K(isAPipe(o)) $\wedge$ K(isABearing(o')) $\wedge$ $K(\neg$isContained(o)) $\wedge$ $K(\neg$isContained(o'))	add(K_f, isHandEmpty(r')), add(K_f, isContained(o')), add(K_f, containsBearing(o)), del(K_f, isRobotGrasps(r', o'))
handoverBearing (robot r, object o, robot r')	K(r $\neq$ r') $\wedge$ K(isRobotGrasps(r, o)) $\wedge$ K(isABearing(o)) $\wedge$ $K(\neg$isContained(o)) $\wedge$ K(isHandEmpty(r'))	del(K_f, isRobotGrasps(r, o)), add(K_f, isRobotGrasps(r', o)), add(K_f, isHandEmpty(r)), add(K_f, $\neg$isHandEmpty(r'))
assembleBearingTree (robot r, object o, object o')	K(isRobotGrasps(r, o)) $\wedge$ $K(\neg$isGrasped(o')) $\wedge$ K(isABearing(o)) $\wedge$ K(isATree(o')) $\wedge$ $K(\neg$isContained(o)) $\wedge$ $K(\neg$isContained(o'))	del(K_f, isRobotGrasps(r, o)), add(K_f, $\neg$isGrasped(o)), add(K_f, isContained(o)), add(K_f, isHandEmpty(r)), add(K_f, containsBearing(o'))
assemblePipeTree (robot r, object o, object o')	K(isRobotGrasps(r, o)) $\wedge$ $K(\neg$isGrasped(o')) $\wedge$ K(isAPipe(o)) $\wedge$ K(isATree(o')) $\wedge$ K(containsBearing(o)) $\wedge$ K(containsBearing(o')) $\wedge$ $K(\neg$isContained(o)) $\wedge$ $K(\neg$isContained(o'))	del(K_f, isRobotGrasps(r, o)), add(K_f, $\neg$isGrasped(o)), add(K_f, isContained(o')), add(K_f, isHandEmpty(r)), add(K_f, isAGearbox(o))

References

Baginski, B. (1997). Efficient dynamic collision detection using expanded geometry models. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 3, pp. 1714–1720.

Bajada, J. (2016). *Temporal Planning for Rich Numeric Contexts*. Phd thesis, King's College London, London, United Kingdom.

Barry, J. L. (2013). *Manipulation with Diverse Actions*. Ph.D. Thesis, Massachusetts Institute of Technology.

Bellingham, J. G., & Rajan, K. (2007). Robotics in remote and hostile environments. *Science*, *318*(5853), 1098–1102.

Bolander, T. (2017). A Gentle Introduction to Epistemic Planning: The DEL Approach. *ArXiv e-prints*, *arXiv:1703.02192*.

Cambon, S., Alami, R., & Gravot, F. (2009). A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, *28*(1), 104–126.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtos, N., & Carreras, M. (2015). ROSPlan: Planning in the Robot Operating System. In *Proceedings of ICAPS 2015*.

Chazelle, B., Dobkin, D. P., Shouraboura, N., & Tal, A. (1997). Strategies for polyhedral surface decomposition: an experimental study. *Computational Geometry*, *7*(5), 327–342.

Cheng, C.-H., Geisinger, M., Ruess, H., Buckl, C., & Knoll, A. (2012). Game Solving for Industrial Automation and Control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4367–4372.

de Silva, L., Pandey, A., & Alami, R. (2013). An interface for interleaved symbolic-geometric planning and backtracking. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 232–239.

Dearden, R., & Burbridge, C. (2013). An Approach for Efficient Planning of Robotic Manipulation Tasks. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 55–63.

Dornhege, C. (2014). *Task Planning for High-Level Robot Control*. Dissertation, Albert-Ludwigs-Universität Freiburg.

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., & Nebel, B. (2009a). Semantic Attachments for Domain-Independent Planning Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 114–121.

Dornhege, C., Gissler, M., Teschner, M., & Nebel, B. (2009b). Integrating symbolic and geometric planning for mobile manipulation. In *Proceedings of the IEEE International Workshop on Safety, Security & Rescue Robotics*, pp. 1–6.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006). Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications, Proceedings of the European Semantic Web Conference (ESEC)*, Vol. 4011 of *Lecture Notes in Computer Science*, pp. 273–287.

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., & Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4575–4581.

Erol, K., Hendler, J. A., & Nau, D. S. (1994). Umcp: A sound and complete procedure for hierarchical task-network planning.. In *AIPS*, Vol. 94, pp. 249–254.

Etzioni, O., Golden, K., & Weld, D. S. (1994). Tractable Closed World Reasoning with Updates. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pp. 178–189.

Ferrer-Mestres, J., Frances, G., & Geffner, H. (2015). Planning with state constraints and its application to combined task and motion planning. In *Proceedings of the ICAPS 2015 Workshop on Planning and Robotics*.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, *2*, 189–208.

Foster, M. E., Gaschler, A., Giuliani, M., Isard, A., Pateraki, M., & Petrick, R. (2012). Two People Walk Into a Bar: Dynamic Multi-Party Social Interaction with a Robot Agent. In *Proceedings of the ACM International Conference on Multimodal Interaction (ICMI)*.

Galindo, C., Fernández-Madrigal, J.-A., González, J., & Saffiotti, A. (2008). Robot task planning using semantic maps. *Robotics and Autonomous Systems*, *56*(11), 955–966.

Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 209–216.

Garrett, C. R., Lozano-Perez, T., & Kaelbling, L. P. (2015). Modular Task and Motion Planning in Belief Space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6366–6373.

Gaschler, A. (2016). *Efficient Geometric Predicates for Integrated Task and Motion Planning*. Dissertation, Technische Universität München.

Gaschler, A., Fischer, Q., & Knoll, A. (2015). The Bounding Mesh Algorithm. Tech. rep. TUM-I1522, Technische Universität München, Germany.

Gaschler, A., Petrick, R. P. A., Giuliani, M., Rickert, M., & Knoll, A. (2013a). KVP: A Knowledge of Volumes Approach to Robot Task Planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 202–208.

Gaschler, A., Petrick, R. P. A., Kröger, T., Khatib, O., & Knoll, A. (2013b). Robot Task and Motion Planning with Sets of Convex Polyhedra. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Gaschler, A., Petrick, R. P. A., Kröger, T., Knoll, A., & Khatib, O. (2013c). Robot Task Planning with Contingencies for Run-time Sensing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Combining Task and Motion Planning*.

Gilbert, E. G., Johnson, D. W., & Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, *4*(2), 193–203.

Gregory, P., Long, D., Fox, M., & Beck, J. C. (2012). Planning Modulo Theories: Extending the Planning Paradigm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Hadfield-Menell, D., Groshev, E., Chitnis, R., & Abbeel, P. (2015). Modular Task and Motion Planning in Belief Space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4991–4998.

Haigh, K. Z., & Veloso, M. M. (1998). Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots*, *5*(1), 79–95.

Hauser, K., & Ng-Thow-Hing, V. (2011). Randomized multi-modal motion planning for a humanoid robot manipulation task. *International Journal of Robotics Research*, *30*(6), 678–698.

Hertle, A., Dornhege, C., Keller, T., & Nebel, B. (2012). Planning with semantic attachments: An object-oriented view. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pp. 402–407.

Hoffmann, J., & Brafman, R. (2005). Contingent Planning via Heuristic Forward Search with Implicit Belief States. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 71–80.

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1470–1477.

Kaelbling, L. P., & Lozano-Pérez, T. (2012). Unifying Perception, Estimation and Action for Mobile Manipulation via Belief Space Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2952–2959.

Kaelbling, L. P., & Lozano-Pérez, T. (2013). Integrated Task and Motion Planning in Belief Space. *International Journal of Robotics Research*, *32*(9–10), 1194–1227.

Karlsson, L., Bidot, J., Lagriffoul, F., Saffiotti, A., Hillenbrand, U., & Schmidt, F. (2012). Combining task and path planning for a humanoid two-arm robotic system. In *Proceedings of the ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications (TAMPRA)*, pp. 13–20.

Kraft, D., Baseski, E., Popović, M., Batog, A. M., Kjær-Nielsen, A., Krüger, N., Petrick, R., Geib, C., Pugeault, N., Steedman, M., Asfour, T., Dillmann, R., Kalkan, S., Wörgötter, F., Hommel, B., Detry, R., & Piater, J. (2008). Exploration and planning in a three-level cognitive architecture. In *Proceedings of the International Conference on Cognitive Systems (CogSys)*, pp. 71–78.

Kress-Gazit, H., & Pappas, G. J. (2008). Automatically synthesizing a planning and control subsystem for the DARPA Urban Challenge. In *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 766–771.

Kuter, U., Nau, D., Pistore, M., & Traverso, P. (2009). Task decomposition on abstract states, for planning under nondeterminism. *Artificial Intelligence*, *173*(5), 669–695.

Lien, J.-M., & Amato, N. M. (2007). Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pp. 121–131.

Mamou, K., & Ghorbel, F. (2009). A simple and efficient approach for 3D mesh approximate convex decomposition. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pp. 3501–3504.

Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, *18*, 87–127.

Nilsson, N. J. (1984). Shakey The Robot. Technical Report 323, AI Center, SRI International.

Petrick, R., Kraft, D., Krüger, N., & Steedman, M. (2009). Combining Cognitive Vision, Knowledge-Level Planning with Sensing, and Execution Monitoring for Effective Robot Control. In *Proceedings of the ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, pp. 58–65.

Petrick, R. P. A. (2006). *A Knowledge-level approach for effective acting, sensing, and planning*. Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

Petrick, R. P. A. (2015). An Application Programming Interface to High-Level Planning with PKS. Tech. rep., School of Informatics, University of Edinburgh.

Petrick, R. P. A., & Bacchus, F. (2002). A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, pp. 212–221.

Petrick, R. P. A., & Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 2–11.

Petrick, R. P. A., & Foster, M. E. (2013). Planning for Social Interaction in a Robot Bartender Domain. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), Special Track on Novel Applications*, pp. 389–397.

Petrick, R. P. A., & Foster, M. E. (2016). Using general-purpose planning for action selection in human-robot interaction. In *Proceedings of the AAAI 2016 Fall Symposium on Artificial Intelligence for Human-Robot Interaction (AI-HRI)*, Arlington, VA, USA.

Petrick, R. P. A., & Gaschler, A. (2014). Extending Knowledge-Level Contingent Planning to Robot Task Planning. In *Proceedings of the ICAPS Workshop on Planning and Robotics (PlanRob)*, pp. 157–165.

Plaku, E., & Hager, G. D. (2010). Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE Intl Conf on Robotics and Automation (ICRA)*, pp. 5002–5008.

Platis, N., & Theoharis, T. (2003). Progressive hulls for intersection applications. *Computer Graphics Forum*, *22*(2), 107–116.

Reif, J. H. (1979). Complexity of the movers problem and generalizations. In *Proceedings of the Annual IEEE Conference on Foundations of Computer Science*, pp. 421–427.

Reiter, R. (1978). On closed world data bases. In Gallaire, H., & Minker, J. (Eds.), *Logic and Data Bases*, pp. 55–76. Plenum, NY.

Rickert, M. (2011). *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

Sander, P. V., Gu, X., Gortler, S. J., Hoppe, H., & Snyder, J. (2000). Silhouette clipping. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 327–334.

Schulman, J., Lee, A., Awwal, I., Bradlow, H., & Abbeel, P. (2013). Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. In *Proceedings of Robotics: Science and Systems (RSS)*.

Srivastava, S., Fang, E., Lorenzo, R., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 639–646.

Srivastava, S., Riano, L., Russell, S., & Abbeel, P. (2013). Using classical planners for tasks with continuous operators in robotics. In *Proceedings of the ICAPS Workshop on Planning and Robotics (PlanRob)*, pp. 27–35.

Sussman, G. J. (1973). *A computational model of skill acquisition*. Ph.D. Thesis, Massachusetts Institute of Technology.

Wolfe, J., Marthi, B., & Russell, S. J. (2010). Combined task and motion planning for mobile manipulation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 254–258.

Xavier, P. G. (2002). Implicit convex-hull distance of finite-screw-swept volumes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 1, pp. 847–854.