



Technische Universität München
Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Efficient Geometric Predicates for
Integrated Task and Motion Planning

Andre K. Gaschler





Technische Universität München
Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Efficient Geometric Predicates for Integrated Task and Motion Planning

Andre K. Gaschler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Thomas Huckle
Prüfer der Dissertation: 1. Univ.-Prof. Dr. Alois Knoll
2. Prof. Oussama Khatib, Stanford University, USA

Die Dissertation wurde am 24.09.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 30.03.2016 angenommen.

Copyright 2015, Andre Gaschler, alle Rechte vorbehalten.

Bildnachweis: Abbildung 1.1, Seite 2, Werk ist gemeinfrei, Digitalisierung von SLUB Dresden unter Lizenz CC-BY-SA 4.0. Abbildung 6.4, Seite 109, mit freundlicher Genehmigung von S. Nogina. Alle anderen Abbildungen wurden vom Verfasser erstellt.

Abstract

“Can’t your robot do this for you?” People have great expectations of what tasks robots can accomplish and have been dreaming of intelligent machines that can understand, perceive, and manipulate. While today’s robot systems may not quite fulfill this dream, research in individual areas on automated planning, path planning, and robot control has made substantial progress. However, integrating these components does not make an intelligent robot. In particular, task planning cannot be solved as a problem separate from motion planning.

This thesis approaches the integrated task and motion planning problem from the geometric side. Its main goal is to develop a powerful and efficient interface for the symbolic task planner to query and sample in a continuous-valued, geometric world. To provide more efficient queries without risking collisions, new geometric predicates are proposed that operate on single-sided, ϵ -precise approximations of the geometry. These single-sided approximations are generated by our new bounding mesh algorithm, which iteratively decimates edges to generate simpler meshes that either enclose or are enclosed by the original geometry. Edge decimations are guided by a quadratic cost function with linear inequalities. Several cost functions are evaluated on a set of robot geometries and further shapes, and experiments indicate that bounding mesh approximation reduces vertex counts by a factor of 10–20 at a good precision. Integration with a convex segmentation algorithm then allows a bounding convex decomposition of the scene, suitable for efficient collision checking routines. Effectively, bounded geometric predicates allow faster collision and inclusion queries, but never overlook a collision or a non-inclusion. Besides these queries, an approach to sampling with geometric constraints is developed to provide a mapping from symbolic preconditions to feasible geometric states. Constraints may be formulated as coincidence, parallelism, and distance relations between shapes of robots and objects, and are solved by projection sampling to cover the constraint space.

All geometric functions are integrated with a knowledge-level automated planner that can reason under discrete uncertainty. In contrast to closed-world planning, discrete uncertainty planning can model gain and loss of knowledge and generate branched plans with run-time sensing actions. The integration leads to the new Knowledge-level Action and Bounding Geometry Motion planner (in short, KABouM), which solves and executes tasks that can be specified by concise domain definitions. The overall approach is demonstrated and evaluated on different robot systems, including tasks with branched plans, sensing actions, assembly actions, and bimanual manipulation. Evaluation includes a complex assembly scenario with seven types of transfer and manipulation actions whose solution requires multi-robot collision checking and synthesis of re-grasping and bimanual manipulation actions, which clearly cannot be solved without hybrid search spaces. Experiments show that bounded geometric predicates increase the efficiency of task and motion planning. Besides direct applications to intelligent robot control, which are illustrated by several robot demonstrations, bounding mesh approximation and bounding convex decomposition apply more generally to real-time collision checking, online control schemes, efficient distance computation, and further algorithms in computer geometry.

Zusammenfassung

„Kann das nicht dein Roboter für dich erledigen?“ Viele Menschen haben hohe Erwartungen an die Fähigkeiten von Robotern, und der Traum von der intelligent denkenden und handelnden Maschine reicht weit zurück. Obwohl die heutigen Robotersysteme diese Hoffnungen nicht ganz erfüllen können, hat die Forschung doch erhebliche Fortschritte in den Teilbereichen der symbolischen Planung, der Bahnplanung und der Robotersteuerung erbracht. Die Integration dieser Komponenten allein liefert jedoch kein intelligentes Robotersystem, da insbesondere die Aufgabenplanung nicht getrennt von der Bahnplanung gelöst werden kann.

Diese Dissertation nähert sich dem Problem der integrierten Aufgaben- und Bahnplanung von der geometrischen Seite. Kernziel ist dabei die Entwicklung einer leistungsfähigen und effizienten Schnittstelle, die der diskreten Aufgabenplanung Abfragen über den geometrischen Zustand und die zufällige Erstellung geometrischer Zustände unter Nebenbedingungen ermöglicht. Für die effiziente Umsetzung dieser Abfragen unter Ausschluss von Kollisionen wird eine neue Art geometrischer Prädikate eingeführt, die durch eine einseitige Approximation der Geometrie definiert ist. Diese Approximation wird durch den neuen Bounding-Mesh-Algorithmus erstellt, der iterativ Kanten dezimiert und vereinfachte Dreiecksnetze erzeugt, die die ursprüngliche Geometrie umschreiben oder darin einbeschrieben sind. Gesteuert wird die Kantendezimierung von einer quadratischen Optimierungsfunktion mit linearen Ungleichungen. Mehrere Optimierungsfunktionen werden auf einer Reihe von Robotergeometrien und anderen Formen experimentell miteinander verglichen. Die Ergebnisse zeigen, dass der Bounding-Mesh-Algorithmus die Zahl der Ecken bei guter Näherung um einen Faktor von 10 bis 20 reduzieren kann. In Verbindung mit einem Algorithmus zur konvexen Zerlegung ist die umschreibende konvexe Zerlegung einer Szene möglich, die sich somit für effiziente Kollisionsabfragen eignet. Damit erzielen die eingeführten geometrischen Prädikate schnellere Kollisions- und Inklusionsabfragen, ohne dass Kollisionen oder Nicht-Inklusionen übersehen werden. Neben

den geometrischen Abfragen wird ein Verfahren zur zufälligen Auswahl geometrischer Zustände unter Nebenbedingungen entwickelt, um eine Abbildung von symbolischen Bedingungen zu geometrischen Zuständen zu schaffen. Nebenbedingungen können in Form von Überschneidungs-, Parallelitäts- und Abstandsbeziehungen zwischen Formmerkmalen in Roboter und Objekt gegeben werden. Zur Lösung werden zufällige Zustände durch iterative Minimierung einer Kostenfunktion auf den Lösungsraum projiziert.

Alle geometrischen Funktionen werden mit einem wissensbasierten symbolischen Planer integriert, der in Anwesenheit diskreter Unsicherheit schlussfolgern kann. Dieser kann Zugewinn und Verlust von Wissen direkt modellieren und somit bedingte Pläne erzeugen, die Laufzeit-Sensorergebnissen entsprechend verzweigen. Die Integration führt schließlich zu dem neuen „KABouM“-System, das kompakt definierte Probleme selbstständig lösen und ausführen kann. Der gesamte Ansatz wird auf verschiedenen Robotersystemen demonstriert und ausgewertet, einschließlich Aufgaben mit verzweigten Plänen und Sensoraktionen, Montageaktionen sowie der Manipulation mit beiden Roboterarmen. Die Auswertung schließt ein komplexes Montageszenario mit sieben Aktionstypen zur Übergabe und Handhabung ein, dessen Lösung die Kollisionsüberprüfung zwischen mehreren Robotern und die Synthese von Aktionen zum Umgreifen und zur beidhändigen Handhabung erfordert, welche daher nur in einem hybriden Suchraum gelingen kann. Experimentelle Ergebnisse zeigen, dass die eingeführten geometrischen Prädikate die Effizienz der integrierten Aufgaben- und Bahnplanung steigern. Neben den direkten Anwendungen zur intelligenten Robotersteuerung, die in mehreren Demonstrationen aufgezeigt werden, sind die Algorithmen zur Umschreibung und umschreibenden konvexen Zerlegung von Dreiecksnetzen in der Echtzeit-Kollisionsverhütung, Echtzeit-Regelung, Abstandsberechnung und in Verfahren der algorithmischen Geometrie einsetzbar.

Acknowledgements

First of all, I would like to thank my adviser Alois Knoll for his guidance through all stages of my study and research work. I am grateful for his constant support, ideas, and the opportunities he has created for me, not only in the field of robotics. He has been encouraging me to new ventures long before I started my doctoral studies.

I am thankful to my second reviewer Oussama Khatib for his valuable advice, for his hospitality, and for giving me the opportunity to conduct several experiments in his lab. Oussama convinced me to develop my approach further in an important way.

Special thanks to Ronald Petrick, Markus Rickert, and Manuel Giuliani, without whom this thesis would not be possible, and to Torsten Kröger for being a competent discussion partner and a friendly host during my stays in Stanford. Many thanks to Mary Ellen Foster, Hauke Stähle, Sören Jentzsch, Quirin Fischer, Ingmar Kessler, and all members of the fortiss robotics group.

Finally, I want to thank my parents and grandparents for their continuous support, and Ursula for being there.

Contents

1	Introduction	1
1.1	Integrated Task and Motion Planning	3
1.2	Scope of this Work	4
1.3	Applications	7
1.4	Contribution	8
1.5	Structure	10
2	Related Work	13
2.1	Background	14
2.2	Related Work in Task and Motion Planning	15
2.3	Task and Motion Planning Systems	18
3	Integrated Task and Motion Planning	25
3.1	Problem Definition	26
3.2	Related Work in Symbolic Planning	28
3.3	Approach to Integrated Task and Motion Planning	30
3.3.1	Planning with Knowledge and Sensing	31
3.3.2	Interface to Robotics-specific Functions	33
3.3.3	FORCE SENSING Scenario	34
3.3.4	Conclusion	38
4	Bounding Meshes for Efficient Geometric Predicates	41
4.1	Bounded Geometric Predicates	42
4.2	Bounding Meshes	45
4.2.1	Level-of-Detail Models	46
4.2.2	Single-Sided Mesh Approximation	48
4.3	Bounding Mesh Generation	50
4.3.1	Bounding Mesh Edge Contraction	51
4.3.2	Quadric Error Metric	52

4.3.3	Quadric Cost for Compound Shapes	54
4.3.4	Optimal Edge Contraction	56
4.3.5	Bounding Mesh Algorithm	60
4.4	Bounding Sets of Convex Polyhedra	67
4.4.1	Algorithms for Convex Decomposition	68
4.4.2	Bounding Convex Decomposition	69
4.4.3	Evaluation	70
4.5	Bounding Swept Volumes	77
5	Sampling with Geometric Constraints	83
5.1	Related Work	85
5.2	Geometric Constraint Formulation	88
5.2.1	Constrained Sampling Problem	89
5.2.2	Design of Geometric Cost Functions	91
5.2.3	Completeness of Constrained Sampling	92
5.2.4	Evaluation of the Sampling Algorithm	93
6	Implementation and Evaluation	97
6.1	System Implementation	98
6.1.1	Geometric Pre-processing	98
6.1.2	Components for Planning and Symbolic–Geometric Mapping	99
6.1.3	Run-time Components	103
6.2	System Evaluation	103
6.2.1	Bimanual Pick-and-Place Scenarios	104
6.2.2	STACKED n OBJECTS Scenario	109
6.2.3	BIMANUAL ASSEMBLY Scenario	115
6.2.4	Conclusion	121
7	Conclusion	123
7.1	Contribution	124
7.2	Future Work	125
7.3	Further Applications	127
A	Technical Definitions and Proofs	129
A.1	Definition of the 3D Quadric Metric	129
A.1.1	Distances to Geometric Primitives	130
A.1.2	Operations on Quadric Metrics	130
A.2	Convexity Invariance of the Bounding Mesh Algorithm	132
A.3	Closed-form Inverse Kinematics	133

B Bounding Mesh Evaluation	135
B.1 Additional Bounding Mesh Examples	136
B.2 Evaluation of Cost Functions for Bounding Mesh Decimation	138
Bibliography	145

Chapter 1

Introduction

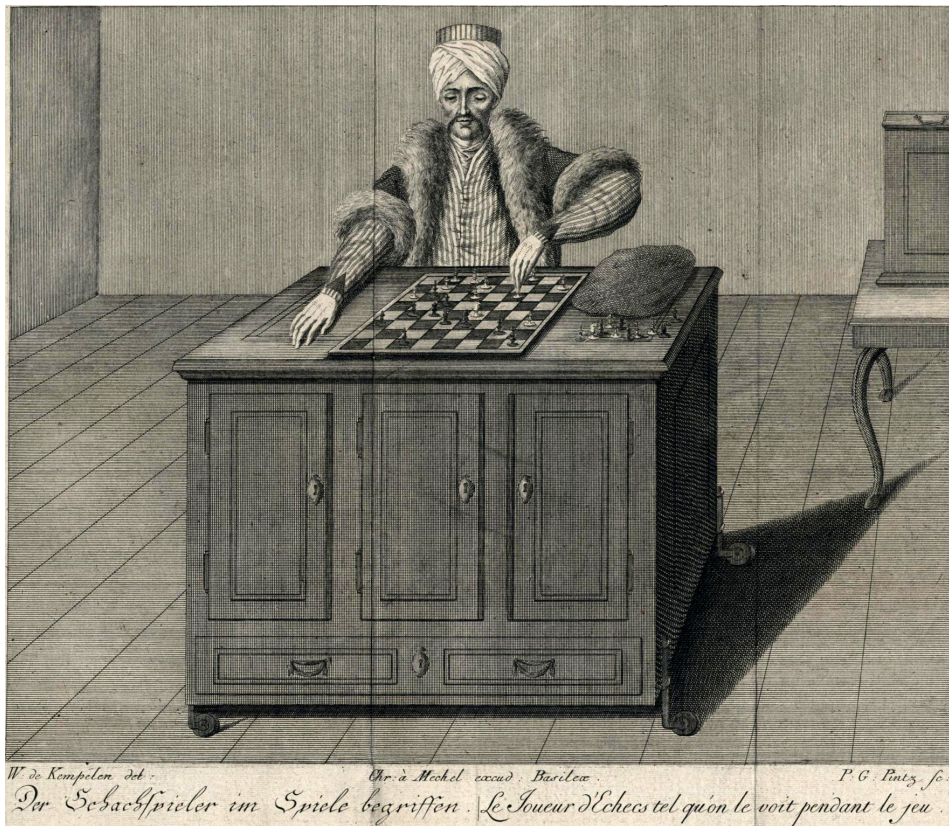


Figure 1.1: “The chess player in the process of playing.” (Der Schachspieler im Spiele begriffen.) Copper engraving by Windisch, 1783, scan by SLUB Dresden, CC-BY-SA 4.0.

The dream of creating an intelligent robot has existed for a long time in human history. The idea of an autonomous, intelligent agent is not only motivated by the prospect of work or service being done by a machine, but is also an interesting field of research. An example of this in history is the “Mechanical Turk” in the late 18th century, constructed by the Hungarian Wolfgang von Kempelen [1]. This mechanism consisted of a humanoid that could apparently move chess pieces on a board and even play strong games against human opponents (Figure 1.1). Even though this “artificial” chess player was only an elaborate illusion, operated by a skilled human player hidden inside the machine, the illusion was at least a visionary one. The audience believed in this intelligent system, a system that has now become feasible with today’s technology in robot manipulation, artificial intelligence, and computer vision. Most of the 18th century audience was fooled by the Mechanical Turk and believed in its abilities to move chess pieces with a mechanical hand and automatically plan strong chess moves. Today’s technology has finally come to a state where

visual object recognition, robot grasping and manipulation, and automated planning are very real, and allows us to build a robotic chess player that picks and places pieces autonomously, which was recently demonstrated by Matuszek et al. [2] among others. Not only was the 18th century audience amazed by the prospects and opportunities of intelligent robot manipulation, people today are also fascinated by intelligent robotics. The Mechanical Turk is an example of a long time technological dream to build an intelligent, humanoid robot with remarkable or even superhuman reasoning and manipulation skills.

The motivation of this thesis is based on the fact that, while most of the technology necessary to build an intelligent robot is now available, it is still surprisingly hard to formulate and solve generic robot problems, and have robots perform useful tasks and services. This thesis seeks to define and solve generic robot tasks, in contrast to a single-purpose chess playing robot, which can be designed without any notion of integrated task and motion planning. Integrated task and motion planning combines symbolic and geometric searches to solve complex, generic problems. Our approach uses robot task and motion planning for solving intricate problems in service and manufacturing. Unlike simple automation tasks, real service and manufacturing scenarios require a robot to perform various types of actions under very complex geometric and kinematic constraints.

1.1 Integrated Task and Motion Planning

The fundamental idea of integrated task and motion planning is to formulate all of a robot's abstract actions together with their geometric constraints and effects in a well-defined and concise language, and then generate plans automatically in a hybrid search. Only in an integrated symbolic and geometric search, a planner can be sure to find viable actions with collision-free paths. With this abstract task and motion planning approach, seemingly complex problems, which typically arise in service and manufacturing scenarios, can be formulated and solved in a generic way. In addition, domain descriptions can be stored separately from the planning system. When the task is defined separately from the planning framework, intelligent robots can be developed that solve the wide range of scenarios that are prevalent in the real world. Since the problem is stated on the task level, generic planners can solve a wide range of problems over different types of robots, tasks, and services. Figure 1.2 illustrates a number of scenarios, all of which can be given as a domain definition to the same planner. These examples include different kinematics from bimanual robots to mobile manipulators and tasks that require sensing and

knowledge-gathering actions.

Another argument for reasoning on the task planning level of abstraction can be made by the way humans think, communicate, and expect robots to interact with them. Humans commonly reason about symbolic actions, knowledge, and geometry in order to fulfill complex, but ordinary tasks; this high-level cognitive function is central to most activities. Furthermore, they can formulate their thoughts to express their knowledge to others or cooperate with others in joint actions. These considerations essentially boil down to two results: First, it is necessary for intelligent robots to perform abstract reasoning and planning on a comparable level in order to operate in a world made by humans. Second, robots that fulfill service or cooperative tasks, or otherwise engage in interactions, are required to understand and express high-level statements in an interaction, formulated in terms of human concepts of geometry and action.

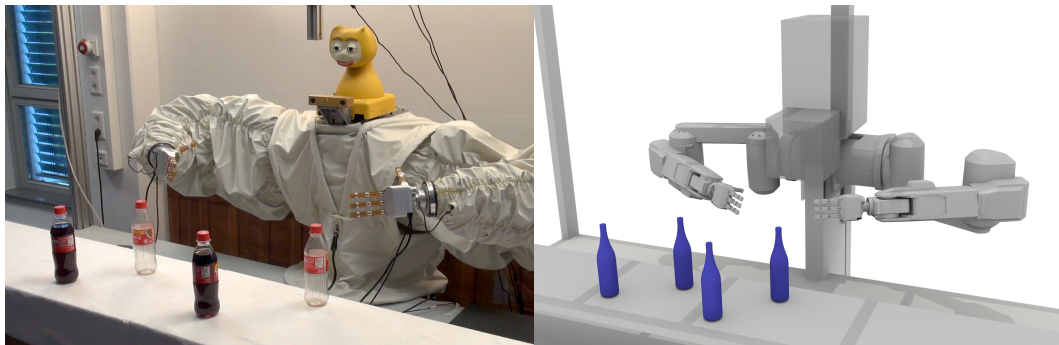
Building a robot task and motion planning system that can plan and act is not a software integration problem. Although such a system requires algorithms from several areas, including symbolic planning, perception, collision checking, path planning, trajectory generation, and execution monitoring, and integration is surely part of the effort, symbolic reasoning and geometric planning must be combined carefully to solve real tasks in a hybrid search. The underlying problem of task and motion planning is that robots need to reason about actions on an abstract, symbolic level in order to achieve a symbolic goal, but these actions have intricate preconditions and effects in the continuous-valued geometric world. Importantly, a two-level architecture, where an automated planner solves a symbolic plan that is later refined by motion planning, will fail in all but the simplest scenarios [3, 4].

1.2 Scope of this Work

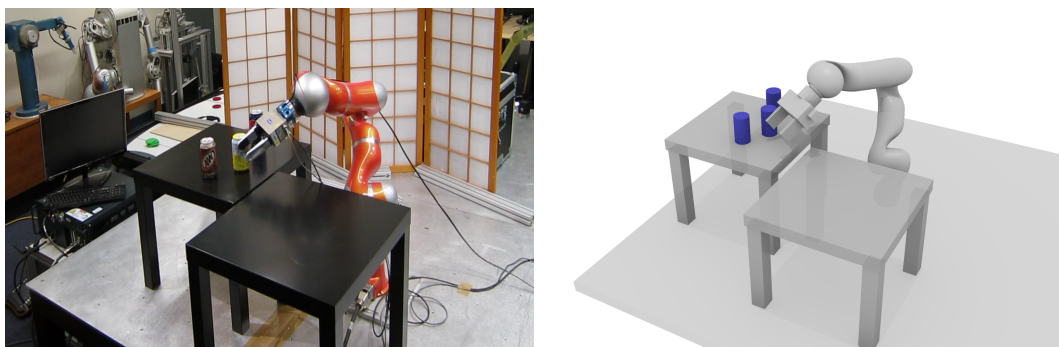
In terms of research questions, this work seeks to answer and is driven by two questions, which are concerned with the geometric side of robot task and motion planning.

Which geometric predicate and sampling functions should motion planning provide to symbolic planning such that real problems can be solved?

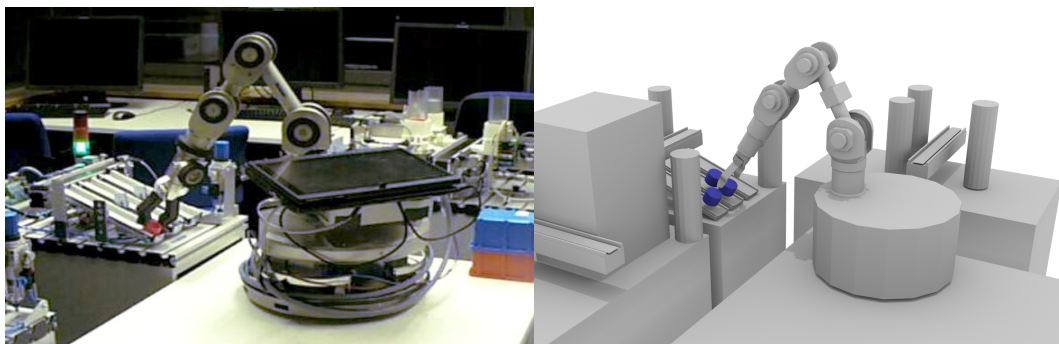
This first question builds on the insight that the symbolic–geometric interface is key to solve real, hard task and motion planning problems [3]. This interface establishes a mapping between symbolic and geometric states. Therefore, the interface both needs to include *predicate* functions that abstract from a geometric state and



(a) Dual-arm MITSUBISHI setup, used in the BARTENDER and BIMANUAL CIRCULAR RE-ARRANGE scenarios.



(b) Compliant LBR4 robot, used in the FORCE SENSING scenario.



(c) Mobile manipulator of a KATANA robot on a ROBOTINO base, used in the STACKED n OBJECTS scenario.

Figure 1.2: A robot task and motion planner accepts a domain definition with actions of symbolic and geometric preconditions and effects, and a symbolic goal. The search algorithm of the planner is independent from the specific domain, and can therefore solve a wide range of problems, including bimanual kinematics and mobile manipulation. A selection of scenarios is evaluated on real robot setups (Chapter 6). In the rendered views, movable objects are highlighted in blue.

resolve symbolic predicates, and *sampling* functions that refine a symbolic state, giving a geometric instance that fulfills symbolic predicates. Thus, the answer to the question about the geometric interface needs to discuss both sides of the symbolic–geometric mapping, provide a set of functions sufficient to solve interesting domains, and demonstrate its effectiveness in real-world examples.

How can geometric predicates in task and motion planning be made more efficient, allowing reasonable approximation of the geometric world?

This second question is concerned with the efficiency of such geometric predicates. Clearly, a robot task and motion planner must cover search spaces larger than a conventional path planner, as the geometric configuration space includes object poses, and may include multiple kinematics. Therefore, the efficiency of a geometric predicate has great impact on the overall efficiency of the planning system. By reasonable approximation, we mean solutions that do not affect the planner’s success to real-world scenarios, either by choosing good heuristics, approximations with controllable precision, or allowing the planner to trade precision for efficiency throughout the search. Furthermore, results of a geometric predicate have different importance to the planner, and approximation may take advantage of asymmetric requirements in precision. An example for such asymmetric tolerance is a collision check, because missing a collision-free path is not as serious as accidentally colliding with an obstacle. A geometric predicate may therefore take advantage of an approximation that guarantees exactness on one side, but may be less precise on the other side. In order to answer this second research question, a framework for geometric approximation is derived that exploits asymmetric tolerances of geometric predicates in robot tasks, and this new type of geometric predicates is implemented and evaluated.

Constraints to the Scope of this Work

This work is concerned with the geometric interface for integrated task and motion planning, in particular with providing efficient geometric predicates. We admit that task and motion planning is a rather vague term. In particular, the word “task” could easily be confused with its different meanings as real-time tasks in robot control, the task space of a manipulator, or motion primitives for task-level robot programming. Our notion of a task planner is an automated planner that generates a sequence of symbolic actions to achieve a given goal in a symbolic domain. To specify the scope of this work more clearly, we would like to mention which topics, although clearly related to intelligent robotics, are *not* within the scope of this work.

Even though integrated task and motion planning clearly involves symbolic and geometric planning, this work is not about developing a new symbolic planner or a new path planning algorithm. Both artificial intelligence and robotics researchers have been developing planners for these two problems for decades, and our contribution is rather to integrate both approaches and develop new geometric predicates to solve more complex scenarios. Furthermore, our approach is different from task-level programming, robot skills, or motion primitives. While these approaches do raise the level of abstraction and implement robot systems that perform multiple actions to achieve a certain goal, they are mostly concerned with reusable and flexible robot programs. In contrast, we attempt a hybrid search in the full space of actions and paths, with tight integration of symbolic and geometric reasoning. Our motivation for integrated task and motion planning is to allow robots plan and act autonomously in a given domain, with no rules given how a task should be fulfilled.

1.3 Applications

The general concept of robot task and motion planning is to provide an abstract problem definition and automatically solve for a sequence of actions for a robot to achieve this task, using generic planning independent from a specific problem. This procedure is designed to solve difficult goals in complex geometries following a task-level description of the domain. It is therefore suitable to applications for service robots, complex manufacturing and construction tasks, and tele-operation of remote robots, where communication is limited to task-level control.

Service tasks require different types of actions, which may include motion, manipulation, sensing, and dialogue with humans. A robot task planner can reason in such a complex symbolic domain, plan actions with intricate dependencies, and solve for sequences of actions that fulfill a goal criterion of multiple conditions. Together with motion planning in complex domains, the ability to coordinate other types of actions, including speech and dialogue, makes robot task planning suitable for developing service robot systems. Motion planning in complex domains is a particular concern in manufacturing and construction tasks, where tolerances are low, space is limited, and manipulators are heavy and stiff. In industrial manufacturing, production needs to become more flexible, provide variants and customization, and minimize programming and setup times. Only when manufacturing processes are described on the task level, automation can achieve this flexibility without user intervention. Robot task and motion planning can automatically generate efficient plans to solve these tasks. Furthermore, it can take advantage of complex re-grasping

actions, bimanual manipulation, and coordinate collision-free motion of multiple robots and objects in a work cell. Clearly, these actions would be too complex or too error-prone to be programmed manually. Besides service and manufacturing robots, planning on the task level also applies to tele-operation of remote robots and robots in space, which need to act autonomously due to limited communication.

1.4 Contribution

In general, we approach the task and motion planning problem from the geometric side. Most authors see the integration of symbolic and geometric planning as the key challenge in robot task planning [3, 5]. Our work is focused on the geometric predicates provided to the symbolic planner; its main contributions beyond the state of the art are the introduction of single-sided approximate geometric predicates and their integration in a knowledge-level planning system.

In particular, we propose a set of geometric predicates that use a new, *single-sided ε -precise approximation*. Single-sided approximation takes advantage of the asymmetric tolerances required for collision and inclusion predicates. Collision predicates evaluate with zero tolerance for collisions, and at a given precision ε in case of non-collisions; the opposite holds for inclusion predicates. As a result, geometric predicates are more efficient without affecting the correctness of the generated plans, which we can show in an experiment. To generate a single-sided approximation of a geometric mesh, we propose the *bounding mesh* algorithm, which may generally be applied to computer geometry. In conjunction with convex decomposition, bounding mesh operations generate a bounding convex decomposition of the scene geometry, which allows efficient evaluation of the above predicates.

Besides this specific contribution of single-sided approximate geometric predicates, we also progress in the integration of task and motion planning systems that handle *discrete uncertainty*. Using an automated planner that operates on the knowledge level, we can model actions where information is gained or lost. This approach provides a well-defined formulation of robot sensing actions, allows reasoning under discrete uncertainty, and can generate plans with branches. Finally, our integration efforts lead us to the new Knowledge-level Action and Bounding Geometry Motion planner (KABouM), and we can demonstrate its effectiveness in a wide range of scenarios. As part of the evaluation, we solve scenarios that require bimanual manipulation, sequences of re-grasps, and complex assembly actions. Furthermore, we demonstrate a selection of problem instances on real robot systems.

Publications

The research leading to this thesis has in part been presented at several conferences and workshops, with the most relevant publications being listed in the following.

- A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, A. Knoll, KVP: A Knowledge of Volumes Approach to Robot Task Planning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 202–208
- A. Gaschler, R. P. A. Petrick, T. Kröger, A. Knoll, O. Khatib, Robot Task Planning with Contingencies for Run-time Sensing, in: IEEE International Conference on Robotics and Automation (ICRA) Workshop on Combining Task and Motion Planning, 2013
- A. Gaschler, R. P. A. Petrick, T. Kröger, O. Khatib, A. Knoll, Robot Task and Motion Planning with Sets of Convex Polyhedra, in: Robotics: Science and Systems (RSS) Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications, 2013
- A. Gaschler, S. Nogina, R. P. A. Petrick, A. Knoll, Planning perception and action for cognitive mobile manipulators, in: SPIE Volume 9025 – Intelligent Robots and Computer Vision XXXI: Algorithms and Techniques, 2014
- R. P. A. Petrick, A. Gaschler, Extending Knowledge-Level Contingent Planning to Robot Task Planning, in: International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (PlanRob), 2014
- A. Gaschler, I. Kessler, R. P. A. Petrick, A. Knoll, Extending the Knowledge of Volumes Approach to Robot Task Planning with Efficient Geometric Predicates, in: IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 3061–3066
- A. Gaschler, Q. Fischer, A. Knoll, The Bounding Mesh Algorithm, Tech. Rep. TUM-I1522, Technische Universität München, Germany (June 2015)

Earlier versions of the KABouM task and motion planner, previously named KVP, the “Knowledge of Volumes” Planner, were described in conference [6, 9] and workshop papers [7, 8, 10, 13], including discussions of the FORCE SENSING and BARTENDER [6] scenarios, and an early version of the STACKED n OBJECTS [9] scenario. In contrast, the more complex BIMANUAL CIRCULAR REARRANGE and

BIMANUAL ASSEMBLY scenarios are novel to this work and the KABouM planner. Concerning the bounding mesh algorithm, a brief technical note was previously published [12] and the idea of single-sided approximation was mentioned in a conference paper [11]. An earlier version of the approach to sampling with geometric constraints was covered by a conference paper [14]. All other work on bounded geometric predicates, bounding meshes, and bounding convex decomposition, including in-depth discussions and evaluation, has not been published before. More detailed remarks on earlier publications are made in the related work discussions of the respective chapters of this work.

1.5 Structure

This thesis is structured as follows. In Chapter 2, we will discuss related works in integrated task and motion planning, as well as relevant approaches from manipulation, motion planning, and further areas. In our discussion, we will situate our approach with respect to related works.

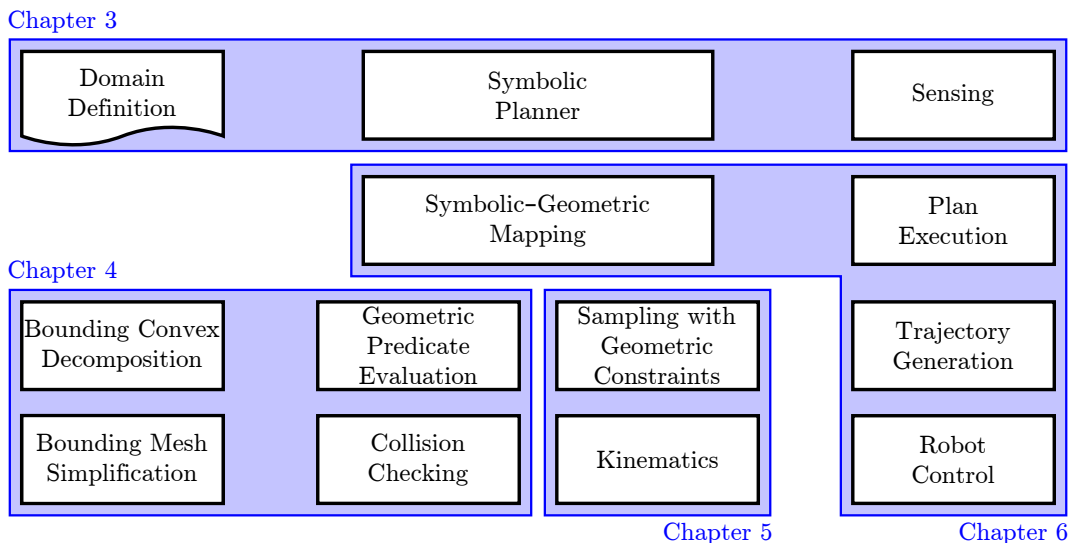


Figure 1.3: Chapter overview in relation to the software architecture of the KABouM planner.

The sequence of the main chapters (3–6) loosely follows our planning system architecture from domain definition to plan execution, as depicted in Figure 1.3. In Chapter 3, we will provide a formal definition of the robot task and motion planning problem. We will then develop our approach starting from the symbolic task planning level, compare automated planning systems, and discuss the solution of a simple

problem that permits separate symbolic and geometric planning. In Chapter 4, we will introduce geometric predicates as an interface to map from geometric states to abstract, symbolic states. To make their evaluation more efficient, we will define a set of bounded geometric predicates on a single-sided, ε -precise approximation of the geometry. We will then derive the bounding mesh algorithm to generate such single-sided approximations of triangle meshes. To complete our set of geometric predicates, we will develop an algorithm for bounding convex decomposition, and provide an efficient collision checking procedure for swept volumes of robot motions. In Chapter 5, we will develop a procedure to sample geometric states that fulfill symbolic conditions. We will formulate a task-level approach to set up geometric constraints, and develop a sampling method that can cover constraint manifolds. To complete the integration, we will describe the components of our new Knowledge-level Action and Bounding Geometry Motion planner (KABouM) in Chapter 6, and evaluate its effectiveness in several scenarios, covering bimanual manipulation, combinatorially intricate rearrangement, and complex assembly tasks. Finally, we will summarize our contribution and discuss possible application areas and future developments in Chapter 7.

Chapter 2

Related Work

For robots to solve real-world tasks, they need to reason about both symbolic actions and continuous-valued paths in the geometric world. In this chapter, we discuss works related to the general robot task planning problem and systems that integrate task and motion planning. Of course, the individual problems of automated planning, geometric predicates, and geometric sampling require more detailed discussion and we will survey the literature on these topics in their respective chapters.

2.1 Background

Research in automated planning started in the late 1960s, at a time when artificial intelligence was sprinting forward and researchers were driven by great optimism to build intelligent machines [15, pp. 18–21]. From the early days on, automated planning was motivated by and applied to controlling autonomous robots. The first intelligent mobile robot, Shakey, was demonstrated to the public in 1969 [16], and could navigate itself, avoid obstacles, and push objects to achieve a given task. Both its automated planner and its pathfinding components were seminal to research in symbolic planning and robot motion planning since then; their combination is considered the first task and motion planning system. The Shakey system included Fikes and Nilsson’s STRIPS planner [17], which is regarded the first major planning system [15, p. 393]. As an automated planner, it solves a symbolic task, defined by a goal criterion, and generates a valid sequence of actions, which fulfill preconditions and entail effects. Using a symbolic planner, the Shakey robot could plan discrete motion, such as moving from one room to another, and respect discrete geometric constraints, such as the connectivity of rooms. After that, symbolic actions were refined by a path planner and executed by the mobile platform. However, the symbolic planner did not check geometric or kinematic constraints, but rather assumed that all actions could be refined to valid motion paths.

Of course, considering kinematic and geometric constraints and effects is crucial for solving the integrated task and motion planning problem. Only in very simple scenarios, symbolic planning and motion planning can be separated; in more general scenarios, a separated search would necessarily become incomplete. In real-world problems, such as problems with movable obstacles, symbolic and geometric constraints tend to be entangled and cannot be solved independently [3]. In 1987, the Handey system was described, which could plan manipulation actions of multiple pick-and-place motions, given a task-level goal and full descriptions of kinematics and geometry [18]. Although the Handey system mainly focused on collision-free path planning, grasping, and manipulation, it also solved a number of discrete prob-

lems in order to accept task-level goals, plan re-grasping actions, choose stable object poses, and coordinate multiple robots [19]. As a task-level manipulation planner, it can be seen as an important milestone towards fully integrated task and motion planning.

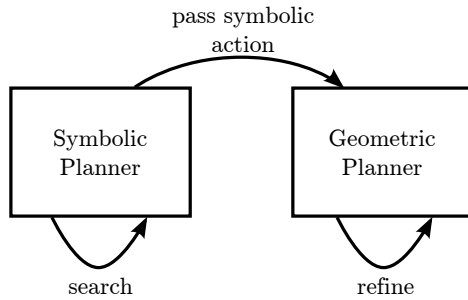
In the 1990s, both the automated planning and motion planning fields made rapid progress, albeit with less interaction between the two fields. Modern algorithms for collision-free path planning were devised to solve complex scenarios by random sampling [20, 21]. However, only in the recent years, the integration of both symbolic and geometric planning has gained new attention and robot task planning has again become an active field of research [3, 22, 23]. One reason for the renewed interest may be that task and motion planning systems have to rely on integrating algorithms and heterogeneous software from different fields, and only recently these algorithms have matured to reusable software components of the necessary planning, control, and vision algorithms [24].

2.2 Related Work in Task and Motion Planning

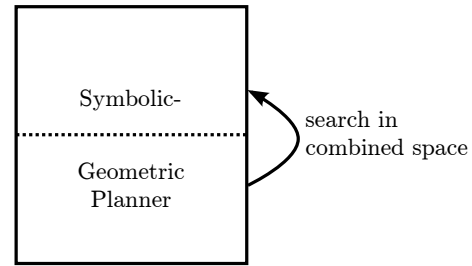
In this young, second era of robot task planning, approaches from diverse research directions have been proposed to combine symbolic and geometric planning. In our discussion of integrated task and motion planning, we first attempt to categorize related works with respect to their general approach, search strategy, and other characteristics. Then, we summarize the most relevant works, roughly following a chronological order. After that, we briefly categorize works from further related fields that describe concepts applicable to task planning. Some techniques from manipulation planning, assembly planning, rearrangement planning, and multi-modal motion planning apply to robot task planning as well, and we try to give a broad overview on the literature.

Survey of Search Schemes

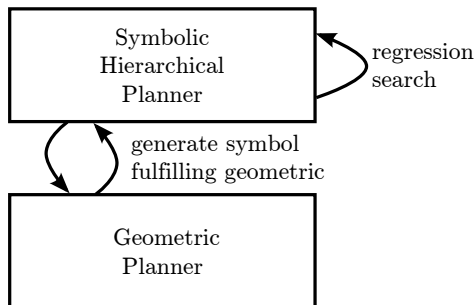
Task and motion planners all perform searches through symbolic and geometric spaces. Search schemes may be progressions, directed forward from a starting configuration to a goal [22, 3, 25, 26], regressions, directed backwards from a goal to the start [23, 27], or may recurse from both start and goal [28]. Some task and motion planners are built on general-purpose automated planners and allow generic domain definitions and multiple types of actions [26, 4, 29, 25], for which the Planning Domain Definition Language (PDDL) is a frequent choice [4, 29]; others use



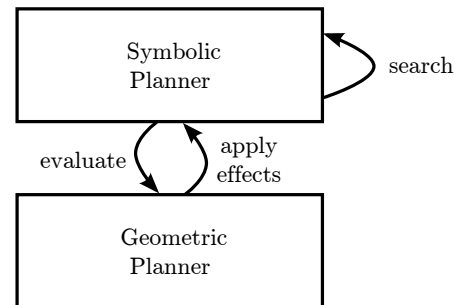
(a) Two-step search, as described by Nilsson [30] for the Shakey system.



(b) Search in the product space, proposed by Gravot, Cambon, and Alami [3] for the aSyMov planner.



(c) Hierarchical backward search with new symbols describing geometric constraints, as proposed by Kaelbling and Lozano-Pérez [31].



(d) Symbols include geometric states, as described by Dornhege et al. [22] and others [26]. Our approach [6, 11] also follows this strategy.

Figure 2.1: Search strategies for combining symbolic and geometric planning. Figure inspired by [31, p. 11].

domain-specific planners, where adding new types of actions would require manual implementation [23, 27].

Hybrid Search Strategies

An important strategy for a task and motion planner is how to explore discrete and continuous search spaces, how to sample in the continuous space, and how to coordinate search in both spaces. In order to categorize search strategies, one can distinguish between symbolic searches that are mostly refined by a geometric planner [32, 16, 26, 33] and geometric motion planners that are guided to fulfill a certain task [25, 34].

Among these many different search strategies, we would like to put a few relevant approaches in comparison. Figure 2.1 gives a schematic overview of four different approaches to integrate task and motion planning. The simplest solution is to plan only symbolical actions and later refine these by motion planning (Figure 2.1a), as in the Shakey system [30]. Gravot, Cambon, and Alami [4] realize that a tighter integration is required to achieve completeness, and search in the product space of discrete and continuous states (Figure 2.1b). Kaelbling and Lozano-Pérez [31] propose a hierarchical planner in the belief space, which can generate new symbols that represent geometric constraints (Figure 2.1c). Several other robot task planners search in the symbolic space and interface a geometric planner to evaluate preconditions and apply effects (Figure 2.1d). The latter strategy is proposed by Dornhege et al. [22]; our KABouM planner and several others [26, 35, 36] also belong to this category.

Hierarchical Planning Strategies

In order to reduce the search space, some approaches propose to plan hierarchically, to interleave planning and execution, or to generate semantic maps. Technically, this leads to different, but related formulations of the robot task planning problem. As opposed to flat planning, hierarchical task networks (HTNs) require a full definition of primitive actions and levels of compound tasks, up to a goal task. The HTN planning approach is taken by the state-abstracted HTN planner [37] and others [38]. Interleaved planning and execution may greatly reduce the search space, since it only needs to solve for the next action. Interleaved planning is proposed by several authors [39, 40], and is suitable for domains with sensor perception, where the robot should react to measurements. It may be complete for domains that contain only reversible actions, among other criteria [40]. On the task level, semantic maps can represent spatial relations and domain knowledge [41].

Incomplete Information and Uncertainty

For robots to plan their actions autonomously, they need to understand that an action may require certain information and may result in gain or loss of information. Only in simple cases, knowledge of information can be expressed by additional predicates in the domain. Some task planners model uncertainty explicitly and can reason with incomplete information [42, 23]. While planning with discrete uncertainty can be achieved for generic domains [42], probabilistic uncertainty requires more domain-specific implementation [23, 27]. Planners that model uncertainty can generate information-gathering actions, such as robot sensing actions, as required

by preconditions of other actions. This contrasts to planners where information-gathering actions can only be defined as tasks themselves. Planning with uncertainty is further necessary to generate contingency plans for actions with execution-time effects.

Applications

Task and motion planning systems are developed with different purposes and applications in mind. Some of the intended applications are control of remote, autonomous robots out of reach of human operators [43], provably correct behavior of autonomous systems [44], and motion generation for the complex kinematics of humanoid robots [34]. Of course, the main area of application is mobile manipulation [4, 22, 23, 35, 26]. Some works are targeted towards service tasks that include interaction with humans [45, 46, 47].

2.3 Task and Motion Planning Systems

Gravot, Cambon, and Alami are among the first to formalize a tight integration of both symbolic and geometric searches [3, 4], with their first version described in 2003 [3]. Their planner, named aSyMov, searches in the product space of both symbolic and geometric variables. The first version of aSyMov [3] is centered on generating graphs for feasible robot–object transfer and robot transit motions in the configuration space. For this, it generates a set of probabilistic roadmaps (PRM), with one roadmap for each object’s location and for each combination of robot manipulator and object locations. Their search identifies intersections between roadmaps, which represent transitions between symbolic states, and finds a hybrid solution through forward search. At each step, the aSyMov planner alternates between searching a viable plan within the existing graph structure or exploring configuration subspaces to increase its geometric knowledge.

In their extended version from 2009 [4], Cambon, Alami and Gravot introduce a more generic formulation of the task planning problem, with problems defined in the more expressive PDDL language. The extended planner uses several heuristics that guide the search algorithm. Geometric search first tries to follow a relaxed, purely symbolic solution, relaxed geometric problems with fewer movable obstacles are solved, and search time is taken away from actions whose geometric evaluation fails often. In their evaluation, they discuss a number of minimalistic, but intricate examples that cannot be solved by independent task and motion planning. As a result, they show that task and motion planning necessarily requires a hybrid

search. In general, Gravot, Cambon, and Alami’s aSyMov planner focuses on the geometric search and uses relaxed symbolic plans mostly as a bias towards the goal. It is shown to be probabilistically complete [4].

Planning with Semantic Attachments

Dornhege et al. [22, 29, 5] also use probabilistic roadmap geometric planning together with symbolic planning. In comparison to aSyMov [3], their geometric planner provides some symbolic information to the high-level planner. For this hybrid interaction, Dornhege et al. define a semantic attachment approach to interface geometric planning. In their definition, a semantic attachment consists of a declarative part in the domain description and a procedural part which is evaluated at planning time. They also distinguish between semantic attachments that check for conditions and those that apply effects, and propose an extension to PDDL to add these features [29]. Erdem et al. [33] describe a causal reasoner that guides a motion planner. They combine high-level reasoning with geometric reasoning by calling external predicates and applying domain-level modifications.

Sampling with Differential Constraints

Plaku and Hager’s Sampling-based Motion and Action Planner (SMAP) [25] is distinctive for it can generate trajectories with differential constraints. Given a set of differential constraints, its sampling routine generates only dynamically feasible robot trajectories. To achieve this, it explores the continuous search space forward in time and propagates geometric states by numerical integration. While SMAP can interact with general-purpose planners on the symbolic level, its search is strongly controlled by a utility function, which guides the search towards under-explored actions and actions part of a discrete solution.

Belief-Space Planning

While the above planners operate on direct symbolic and geometric world states, Kaelbling and Lozano-Pérez devise an approach to plan in the *belief space*, defined as the space of probability distributions over world states [23, 27]. With this probabilistic state representation, they can not only express continuous-valued uncertainty about the outcome of an action, but also uncertainty about the current state. In order to reason in this belief space, Kaelbling and Lozano-Pérez derive a new set of fluents that can characterize preconditions and effects of the robot’s belief of the world state (as opposed to the exact world state), and a set of action definitions

suitable for mobile manipulation domains. Since sensing actions add certainty to the belief state, the planner intrinsically chooses them where necessary, and sensing actions need not be hard-coded as tasks themselves. The intention of their planning architecture, named Belief-space Hierarchical Planning “in the Now” (BHPN), is to solve robot tasks that involve sensing and navigation in a domain with uncertain and incomplete information, with the robot to act autonomously over large time scales.

Of course, complete sampling of the belief space would be prohibitively expensive, especially over longer horizons with many objects. For this reason, they take a strongly hierarchical approach and plan “in the now”; the BHPN planner generates an aggressively hierarchical plan, and directly executes the first low-level robot action. After executing an action, the belief state is re-estimated and the planner refines the abstract plan, following an interleaved planning–execution scheme. Kaelbling and Lozano-Pérez argue that this extremely short-horizon search is a powerful heuristic for uncertain and large domains, and they can show the completeness of the search under a condition that actions are reversible. Contrary to generic task planners that reason on world states [4, 22], BHPN’s architecture and its belief space formulation requires the user to specify regression functions and heuristic generator functions, which depend on start and goal states of a subtask. BHPN performs a regression search, following a pre-image back-chaining strategy, which is rather distinctive among robot task planners. Unlike approaches that use general-purpose symbolic planners [4], BHPN is not designed as a conventional automated planner, and adapting it to a different domain would not be straightforward. In general, BHPN makes several interesting contributions to robot task planning: First, it uses a belief-space formulation of a mobile manipulation domain, where sensing actions are implicitly used to gain useful information and to solve tasks. Second, it takes a strongly hierarchical, interleaved planning and execution approach, potentially allowing large-scale domains.

Leviñ et al. [48] improve the efficiency of the BHPN planner and the quality of generated plans by adding mechanisms for adaptive replanning and foresight towards future subgoals. In a recent work, Hadfield-Menell et al. [49] obtain deterministic representations of belief space planning problems using maximum likelihood estimation. They can integrate their system with an off-the-shelf automated planner and plan with uncertain continuous-valued observations. In contrast to BHPN, their approach is limited to deterministic actions.

Manipulation Tasks

In the course of the GeRT project on manipulation task planning, several approaches to hybrid planning were proposed. Leidner et al. [50, 35] integrate a symbolic-geometric search with backtracking, including an operational space manipulation controller [51] to solve mobile manipulation tasks that require force control and whole-body motion.

Dearden and Burbridge [32] propose a symbolic task planner with geometric refinements, following a strongly symbolic approach where symbolic predicates express geometric relationships. Contrary to fully hybrid planners that try to find plans in the complete search space [4, 22], their work is an interesting example of how a mostly symbolic search with an incomplete geometric evaluation can solve challenging manipulation problems.

Another way to prune the geometric state space is to extract sets of constraints automatically from symbolic actions and evaluated geometric properties, a technique proposed by Lagriffoul et al. [52]. Rather than performing complete geometric backtracking on single geometric states as most planners do [3, 22], they generate linear constraints for ranges of symbolic and geometric states. In a preliminary demonstration, they can automatically generate intervals of feasible grasping angles for a known type of object, and effectively prune the search tree for this type of action.

Hierarchical Planning

Karlsson et al. [38] use hierarchical task network (HTN) planning, with geometric suggester functions to generate new geometric states. Their demonstration includes a pick-and-place task on a humanoid robot. Wolfe, Marthi, and Russell [37] propose the State-Abstracted Hierarchical Task Network planner (SAHTN), which combines symbolic planning with sampling-based kinematic optimization. Their planner creates a global cache with actions as keys and state vectors as values. With this cache, queries can re-use cached results that differ only in values irrelevant to that action. They can show that hierarchical optimality is preserved by this caching strategy.

Propagation of Geometric Failures

In a recent work, Srivastava et al. [26] present a robot task planning approach that relies on symbolic explanations for failures in the geometric search. Their underlying principle is that geometric predicates, such as the absence of collisions, can only be evaluated for individual instances of the geometric search space, but the reason for a negative evaluation of geometric predicates can be explained symbolically, and then

guide further search. The main limitation of their approach is that failures are not always trivial to explain; other geometric predicates, such as the visibility queries used by de Silva et al. [45], would be impractical to explain on the symbolic level.

In a first step, Srivastava’s algorithm invokes a general-purpose planner, assuming all geometric preconditions to be met, and generates a high-level plan. This symbolic plan is then refined by a geometric search; importantly, geometric failures are formulated as symbolic predicates, and geometric choices are discretized to new symbols, both of which are added to the symbolic state. Their geometric refinements use several heuristics that are common in path planning with movable obstacles. As an example, their path planner tries to avoid collisions with any movable objects before resorting to paths that collide with an object. Srivastava et al. prove that their search is complete under an assumption that only predicates of equal signs appear in the goal criterion and in action preconditions and effects, with a formal explanation given in [26]. As an example, this assumption intuitively holds for pick-and-place domain definitions, in which predicates for non-collision and reachability appear with a positive sign. Since their task planning system makes relatively weak assumptions about the symbolic planner, it can be integrated with different general-purpose planners.

Manipulation Planning

Robot manipulation planning is clearly related to integrated task and motion planning, with several manipulation planners attempting to solve tasks by multiple primitive actions [34, 28]. In particular, multi-modal manipulation planners can generate paths over multiple contact states, and transit and transfer actions.

Hauser and Ng-Thow-Hing [34] propose the randomized multi-modal motion planner (Random-MMP), which samples and searches in a hybrid search space of continuous configurations and discrete contact states. Random-MMP can find discrete mode switches (such as contact state changes) in the search space, and considers kinematic and dynamic constraints. In a demonstration, Random-MMP plans a motion path for a full-body humanoid to walk and push an object on a table, which involves many contact changes. While not reasoning on the task level, some manipulation planners have become efficient at solving problems that require many different primitive actions, similar to the problems solved by task and motion planners. One such diverse-action manipulation planner is proposed by Dogar and Srinivasa [53], which models the uncertainty of object poses and effects of actions. In a recent thesis, Barry [54] describes the diverse action manipulation planner (abbreviated, DARRT), which also solves multi-modal paths. Similar to Random-MMP [34], it

samples in the product space of robot and object poses. Its search algorithm follows a generalization of bidirectional rapidly-exploring random trees, with specific projection functions to find lower-dimensional crossings between actions. DARRT can solve manipulation tasks with multiple movable objects and types of actions, including limited usage of tools.

Assembly Planning

Another motivation to study collision-free motion sequences of several objects is assembly planning, with the objective to increase the efficiency of computer aided design for assemblies of parts. With its research being most active in the 1990s, assembly planners are among the first algorithms to understand multiple-object motions and their preconditions as symbolic graph structures, and to identify object relations symbolically. As an example, Halperin, Latombe, and Wilson [55] segment the possible motion space into regions with equal geometric constraints among a subset of parts, or subassembly. With this approach, they can generate a symbolic graph to represent which geometric constraints will be changed by which action. Even though assembly planning is more closely related to multi-robot motion planning than to automated planning, it is interesting to note how the discretization of geometric states is key to several assembly planning algorithms.

Summary

In the recent decade, the field of integrated task and motion planning has made rapid progress after the presentation of the first hybrid planner, aSyMov [3], and a wide range of search techniques has been proposed [4, 37, 22, 23, 38, 35, 26]. Besides planning in the product space of symbolic and geometric states [3, 4], approaches to the related problems of hierarchical planning [38, 37] and belief-space planning [23, 27] were proposed. Similar to most authors, our motivation to integrated task and motion planning is driven by the fact that complex scenarios can only be solved by a hybrid search; our application is targeted towards robot manipulation planning, including bimanual operations and assembly. Considering the different search schemes discussed in Section 2.2, our search is a progression by an off-the-shelf symbolic planner, where geometric preconditions and effects are evaluated on demand, comparable to the semantic attachment approach [22]. Note that our system integrates with a general-purpose planner that can reason under discrete uncertainty [6]. Reasoning with uncertainty is otherwise offered by problem-specific planners [23]; only recently, Hadfield-Menell demonstrated an off-the-shelf planner to handle uncertain observations [49]. In contrast to other approaches, our KABouM planner

operates on a single-sided approximation of the geometry for greater efficiency. This approximation by bounding meshes is novel to our approach, and will be elaborated in later Chapter 4.

In the following, we will provide a formal definition of the task and motion planning problem. After that, we will describe our planner in all details, starting from high-level symbolic planning to geometric predicates, sampling with constraints, and implementation.

Chapter 3

Integrated Task and Motion Planning

The main intention of integrated task and motion planning is to enable robot systems to perform useful tasks in the real world. For robots to operate in complex domains such as service, manufacturing, construction, or tele-operation, they need to reason about both symbolic actions and the geometric world. The area of robot task planning is concerned with solving these types of high-level tasks, which cannot be achieved by simple control schemes, by applying a number of actions that each make predictable changes. From a very broad view, the general approach of task planning is to solve for a sequence of simple actions that the robot can execute in the world, such that the world finally fulfills certain goal criteria. While these actions are defined as manipulation or sensing actions whose result can easily be planned ahead, they may only be applicable under a combination of symbolic, kinematic, and geometric conditions. Only for the most simple tasks, symbolic planning can be separated from motion planning; for solving more useful tasks, an integrated symbolic and geometric search is required [56].

3.1 Problem Definition

In the following, we define the formal *integrated task and motion planning problem*, with the notation that will be followed throughout this thesis. In the most general form, an integrated task and motion planning problem instance is a 3-tuple $(\Sigma, \mathcal{R}, \mathcal{M})$, where Σ is the purely symbolic domain, \mathcal{R} the kinematic definition of all robots, and \mathcal{M} the geometric definition of all robots and objects. Each of these structures is a composition of simpler structures, which are defined as follows.

The symbolic domain Σ is a 4-tuple $(\mathcal{S}, \mathcal{A}, I, G)$ of a set of symbols \mathcal{S} , a set of actions \mathcal{A} , an initial symbolic state I , and a set of goal criteria G . Essentially, Σ represents an automated planning domain of first-order predicate calculus, similar to the original STRIPS formulation by Fikes and Nilsson [17], with slight additions towards sensing and manipulation actions. \mathcal{S} defines the set of discrete symbols that may appear in symbolic definitions. The set \mathcal{A} includes all actions; an action $\mathbf{a} \in \mathcal{A}$ is again a tuple $(\mathbf{p}, \text{pre}(\mathbf{a}), \text{eff}(\mathbf{a}))$, composed of a set of parameters \mathbf{p} , a list of queries as preconditions $\text{pre}(\mathbf{a})$, and a list of effects $\text{eff}(\mathbf{a})$. The parameter set \mathbf{p} is a set of variables that the planner must instantiate to evaluate the action, and which may appear in the preconditions and effects of that action. A precondition can either be an atomic formula querying the symbolic state, or a binary evaluation of a kinematic or geometric query. (The list of kinematic and geometric predicates will be defined in later Chapter 4.) A list of preconditions $\text{pre}(\mathbf{a})$ is then a conjunctive set of preconditions that must evaluate true for an action instance to be applicable

to the planning state. An effect either adds or removes a simple statement in the discrete state, or calls a kinematic or geometric function that makes a change to the continuous-valued state. The goal criteria G are likewise a conjunctive set of conditions as the preconditions are, but do not require any parameters. In particular, goal criteria allow both symbolic and geometric queries.

The set of kinematic structures \mathcal{R} contains one element $\mathbf{R} \in \mathcal{R}$ for each robot in the domain. A kinematic structure \mathbf{R} is further a tuple $(s, \text{FK}, \mathbf{q}_0)$ composed of an associated discrete symbol s , a forward kinematic function FK and an n -dimensional initial configuration vector \mathbf{q}_0 . The initial configuration \mathbf{q}_0 represents the initial angles and lengths of revolute and prismatic joints in the kinematic, respectively. The forward kinematic function FK is a mapping $\mathbb{R}^n \times [0..n] \mapsto \text{SE}(3)$ from the configuration space and a rigid body identifier to the operational space of that rigid body. Typically, FK is only defined for a part of the configuration vector space, depending on the ranges of the joints.

The geometric world is defined by a set of geometric models \mathcal{M} . Each geometric model $\mathbf{M} \in \mathcal{M}$ is a tuple $(s, \mathcal{B}, \mathbf{x}_0)$ of an associated symbol s , analogous to the kinematic structure definition, a set of geometric meshes \mathcal{B} , and a 3D pose $\mathbf{x}_0 \in \text{SE}(3)$. A set of meshes is further composed into mesh elements $\mathbf{B} \in \mathcal{B}$ that may be given as sets of vertex triples, and which describe the geometric boundary of a rigid body of a robot, or of an object in the domain. A vertex triple is a 3-tuple of vertices $(v_0, v_1, v_2) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$. All meshes are constrained to be closed, orientable 2-manifolds. Furthermore, it is assumed that vertices are enumerated counter-clockwise, i.e. $(v_1 - v_0) \times (v_2 - v_0)$ points outside. For many objects in a domain, the geometric model will contain a single mesh; in case of an n -degrees-of-freedom robot, the geometric models typically contains $|\mathcal{B}| = n + 1$ meshes, one for each link. While this concludes the problem instance definition, slight syntactical additions may be introduced by the symbolic planning system. Depending on the automated planner in use, action definitions may allow more general constructs for quantified reasoning or numerical expressions. In addition, the interface to the geometric domain through non-symbolic predicates and effects is further clarified in Section 3.3.2.

The integrated task and motion planning problem is to solve a viable plan for a given problem instance $(\Sigma, \mathcal{R}, \mathcal{M})$ as defined above. In this respect, a plan is a sequence or tree of actions and action parameters that, subsequently applied to the initial state $(I, \mathbf{q}_0, \mathbf{x}_0)$, will end in a state that fulfills all goal criteria G . In particular, the actions in the plan must in all steps fulfill all preconditions and apply their respective effects. The most important extension to the traditional STRIPS

formulation [17] and classical, artificial intelligence planning is that many of these preconditions depend on (and many effects make changes to) the non-symbolic state, whose structure is defined in \mathcal{R} and \mathcal{M} . In the case of a branched plan, where a tree of actions is generated, and unknown variables exist that are resolved at run-time and decide which branch is to be taken, all leaves of the plan must fulfill the goal criteria.

3.2 Related Work in Symbolic Planning

In the field of artificial intelligence, the *automated planning* problem is to generate a tractable sequence of actions that achieves a certain goal, given an action schema of the preconditions and effects of these types of actions [15, Ch. 10]. (Other authors refer to actions as *operators* and to classical planning as logic-based *discrete planning* [57].) The classical planning problem is formally related to the Boolean satisfiability problem, to which it can be translated when the set of symbols is finite. While finding a satisfying Boolean assignment is a viable approach and comparably good at solving hard instances, practical planning instances are more suitable for heuristic searches. In general, the search in the state space can be directed forward (progression search) or backward (regression search). Both search techniques must handle the exponential growth of the state space with respect to the number of actions. In some scenarios, backward search can use partially uninstantiated actions and keep the branching factor lower, but still requires further heuristics for good efficiency. The general approach to a heuristic search is to apply domain-independent relaxations. Standard ways to relax actions are to ignore some or all of their preconditions, or to ignore effects that delete literals, as proposed by Hoffmann and Nebel [58]. Other heuristics involve abstracting states by ignoring fluents or decomposition of the goal into independent subgoals [15]. Another popular approach is the generation of a *planning graph*, which is a subset of the full state transition graph and allows reachability analysis to guide the search. Planning graphs were first described by Blum and Furst [59].

In contrast to traditional planning representations, more expressive languages were proposed and investigated. An example that is relevant towards our application of robot manipulation and sensing is the situation calculus language, as introduced by McCarthy [60] and refined by Reiter [61]. Situation calculus is designed to model linear time with branching situations, and exhibits some features of a second-order language. As an example, the action language Golog, which is based on the situation calculus, has been applied to multi-robot task planning [62]. However, in situation

calculus planning, efficiency to solve practical problems is considerably sacrificed for expressiveness of problem definitions [15, p. 388]. Other generalizations include temporal planning, where actions require a certain time, concurrent planning, where multiple actions may be performed simultaneously, or probabilistic planning, where the state can only be observed up to an uncertainty.

With our application to robot task planning in mind, we limit the discussion to problem formulations that can efficiently be solved. We therefore focus our discussion on high-level task planning with deterministic, non-concurrent, sequential actions with only discrete uncertainty that is planned centrally for all robots. Besides the already mentioned STRIPS formulation for domain descriptions, which was proposed by Fikes and Nilsson in 1971 together with the first major planning software [17], PDDL [63] has more recently gained popularity, especially through the International Planning Competition and wide acceptance among planning software. PDDL separates the domain description with its action schema, predicates and effects definition from to problem instance, which includes initial and goal states of a particular instance. Its core syntax is stable and the most common format for software planners. In addition, many extensions and variants were proposed to accommodate more expressive features, for instance numerical fluents and multiple agents.

Automated Planning Software

Among planning software systems, several implementations have been applied to robotics tasks, on which we focus our discussion. The Fast Forward planner (FF) by Hoffmann and Nebel [58] was the first to combine a hill-climbing search with a goal distance heuristic that ignores fluent-deleting effects. The heuristics of FF and its variants have proven particularly efficient for practical benchmark problems [15, p. 395]. Similar to FF, the Fast Downward (FD) planner, developed by Helmert [64], progresses the search following several heuristics. However, it translates the problem to a multi-valued planning task representation instead of operating in a conventional propositional representation. The heuristics of the FF planner have been adapted to work in a robotics task and motion planning system for multi-object manipulation [65]. With careful interface implementation and symbolic mapping of continuous variables, multi-object robot manipulation can directly use domain-independent planners, such as FF and the cost-sensitive FD planner, as demonstrated by Srivastava [66]. Other planners are dedicated to hierarchical task planning, where larger-scale domains can be handled when a domain-specific hierarchy of actions is provided, instead of a classical STRIPS domain (Section 3.1). An implementation

that has been applied to robot task planning is the Simple Hierarchical Ordered Planner (SHOP), which performs ordered task decomposition. SHOP was described and made available by Nau et al. in 1999 [67], and has been applied in several hierarchical task and motion planning systems since, for instance Bidot’s forward-chaining combined task and path planner [36].

While the above-mentioned task planners apply general-purpose, artificial intelligence (AI) planners, several integrated task and motion planning systems use domain-specific planners, and many of the discrete search schemes directly encode a fixed action schema rather than separating the domain description from the code. As an example, Kaelbling and Lozano-Pérez’ belief-space hierarchical planner “in the now” (BHPN) [23, 27] includes a domain-tailored backtracking search that directly evaluates actions, implemented in a scripting language. One of the first hybrid task and motion planning systems, the aSyMov planner by Gravot, Cambon, and Alami [3] included a hand-crafted search, and was later revised to a version that can parse more generic domain descriptions [4].

3.3 Approach to Integrated Task and Motion Planning

Our approach to the integrated task and motion planning problem (Section 3.1) is built on a number of design principles. In short, these principles are concerned with the separation of search and domain description, the combination of discrete task space and continuous geometric configuration space, and features necessary to solve challenging scenarios in robot applications.

- *General AI planning:* We apply a general-purpose, automated planner and define the search domain, which varies between robots and robot applications, separate from the planner. This way, our task planner can leverage domain-independent heuristics that are implemented in state-of-the-art AI planners, and may benefit from future improvements in the field of automated planning.
- *Bounded geometric predicate and geometric sampling interface:* There is a wide consensus [4, 27, 66] that all but the most trivial robot tasks can only be solved in a hybrid search, where a task planner can generate geometric samples, evaluate geometric preconditions, and refer to geometric effects or new geometric states through discrete-geometric state mapping. Providing novel, efficient geometric queries is one of the main contributions of this work. Our approach is to implement geometric queries as *bounded geometric predicates*, which are made particularly efficient through single-sided, ε -precise geometric

approximation. Their definition and implementation are elaborated in Chapter 4.

- *Planning with discrete uncertainty and sensing actions:* Intelligent robots in service and manufacturing services need to reason how to obtain information that is required to solve a task. With moderate generalizations of the classical STRIPS formulation, discrete uncertainty as well as gain and loss of information can be modeled [68, 69], which is necessary to solve these types of robot scenarios.

Concerning the task planner, these design principles lead to the choice of the Planning with Knowledge and Sensing (PKS) planner by Petrick and Bacchus [68, 69, 70] as a general-purpose planner in our KABouM system. Other aspects of our approach, such as the choice of swept volumes as an intermediate representation for robot motion, which can be both referenced in symbolic and geometric reasoning, are discussed in Chapter 4. The discussion of our KABouM planner is directed from the high-level task planner towards lower-level geometric predicate evaluation, followed by notes on the implementation and several experiments. In the following section, we describe the symbolic planner PKS, along with the task planning scenario FORCE SENSING. While this scenario features discrete uncertainty, which motivates the discussion of conditional planning of robot sensing actions, it is kept intentionally simple as it does not require geometric queries.

3.3.1 Planning with Knowledge and Sensing

Planning with Knowledge and Sensing (PKS) is a conditional automated planner that describes a state by the agent’s knowledge about the world, rather than the world state itself. It was first presented by Petrick and Bacchus in 2002 [68] and has further been developed by Petrick [69, 70]. Its formal language extends the classical formalism of the Stanford Research Institute Problem Solver (STRIPS) for expressing an agent’s knowledge of the world. Contrary to classical planning, PKS does *not* assume a closed world, where all predicates are either true or false. The agent’s knowledge of the world may be incomplete, and the agent (or, robot) may gain or lose knowledge as an effect of performing an action. This knowledge-level approach effectively allows planning with discrete uncertainty and is therefore well-suited for robot task planning.

Knowledge-level Statements

In order to express an agent’s knowledge of a predicate $P(x)$, PKS introduces the additional modal operator K to provide the expression $K(P(x))$. Contrary to closed-world planning, this allows the statements $K(P(x))$ for “ $P(x)$ is known to be true”, $K(\neg P(x))$ for “ $P(x)$ is known to be false”, and $\neg K(P(x))$ for “ $P(x)$ is not known”. While the operator K is most commonly used in all preconditions, two additional operators K_w and K_v are available to express knowledge of a value that will become known at *run-time*, when the robot actually performs actions in the physical world. K_w expresses the knowledge of a binary value that will be resolved when the plan is executed. While the value of a predicate $P(x)$ may be unknown at the time of planning, which is the case for sensor measurements, the expression $K_w(P(x))$ can already be used as a planning-time effect of such a sensing action, and in preconditions of actions that require its measurement result. In this case, PKS generates a conditional plan that provides two branches, one for each outcome of the run-time sensing action. Analogous to the binary knowledge of K_w , the operator K_v expresses the knowledge of a function value $f(x)$ at run-time, where f is an unnested function term. The literal $K_v(f(x))$ can likewise be used in effects of sensing actions and preconditions of actions. PKS offers two further operators to express disjunctive and local closed-world information, however, only the three previously mentioned types of knowledge are used in this work.

Even though all formulas with the operators above, including K_w and K_v , could be translated into formulas with the modal operator K [70, p. 181], PKS separates formulas by the type of knowledge and stores them in different databases, named \mathcal{K}_f , \mathcal{K}_w and \mathcal{K}_v . Regular fluent knowledge, declared with the operator K , is stored in a database \mathcal{K}_f , while the types of knowledge with operators K_w and K_v are stored in the databases \mathcal{K}_w and \mathcal{K}_v , respectively. State updates, which add or delete knowledge and are part of an action’s effects, must refer to which database a formula is added to or deleted from. Separating the different types of knowledge, together with some limitations on the possible types of queries, allows a very efficient implementation of an inference algorithm [70, p. 187].

Knowledge-level Queries

To allow reasoning on this extended, knowledge-level state of the world, PKS also provides an extended set of primitive queries. For an atomic formula ϕ , primitive queries are available to answer whether ϕ is known to be true $K(\phi)$ or false $K(\neg\phi)$, or if the robot will know the binary value of ϕ at run-time $K_w(\phi)$. To query whether the robot will know the value of an unnested function term f at run-time, a primitive

query $K_v(f)$ is provided [70, p. 186]. Besides these primitive queries, their negation is also permitted. PKS chooses this set of primitive queries to both support a wide range of planning problems with incomplete information, and ensure efficient inference from the knowledge databases. Queries are employed at several places of a planning problem definition: The set of preconditions $\text{pre}(\mathbf{a})$ of an action \mathbf{a} is a conjunction of primitive queries. Furthermore, the effects of an action, $\text{eff}(\mathbf{a})$, are formulated as conditional updates to the state of knowledge. An effect may optionally define a conditional primitive query, and always specifies a formula to be added to or to be deleted from one of the databases.

Similar to classical planning in world states, the knowledge-level PKS planner makes assumptions about the knowledge state of the world: It generally requires that the agent has complete knowledge of effects and non-effects of actions, and that the agent’s actions are the single source of changes to the world [70, p. 199]. However, more general scenarios can be solved when replanning is allowed [71]. (Of course, actions for multiple robots can always be planned when planning and execution is centralized.)

Note that these assumptions are different from hierarchical planners that allow later refinement of actions. As an example, Kaelbling and Lozano-Pérez’ hierarchical planner “in the now” interleaves planning and execution, expects actions to fail, and refines abstract actions to reconstruct a working plan [40], and its successor BHPN [27] models effects as changes to the agent’s belief, given as a probability distribution over world states.

3.3.2 Interface to Robotics-specific Functions

Apart from these purely symbolic queries, the Planning with Knowledge and Sensing (PKS) planner can call a domain-specific function $g(\mathbf{x})$ outside the symbolic planner, which can serve as an interface between symbolic task planning and continuous-valued kinematic and geometric motion planning. When encountering such an external function, PKS can pass over a list of symbols \mathbf{x} as an argument list to such a library function g , receive its result, and resume symbolic progression. The given list of symbols \mathbf{x} directly maps to the symbols in the knowledge state and therefore provides a generic interface between the symbolic planner and robotics-specific functions. In the current implementation, PKS can interact with a domain-specific function $g(\mathbf{x})$ through the symbolic–geometric interface in three different ways:

- Parts of the symbolic state can be passed to the robotics-specific layer through the list of symbols \mathbf{x} . Since conjunctions are guaranteed to be short-circuit

evaluated, a conjunction $K(\phi) \wedge g(\mathbf{x})$ can conditionally pass knowledge to the interface.

- The robotics layer can be queried through a domain-specific function and effectively add or delete symbolic knowledge. To achieve this, the function g may be placed in the conditional part of an effect to update a specific database. For this, effects of the form $g(\mathbf{x}) \Rightarrow \text{add}(\mathcal{K}_f, \phi)$ are allowed. Likewise, updates to the other databases and deletion of knowledge are allowed.
- The result of g may be compared to the knowledge state through the nested query $K(\phi = g(\mathbf{x}))$. This is currently the only nested query allowed in PKS, and future work may expand the set of possible queries.

From the planner’s point of view, a domain-specific function g can be an arbitrary software library function, with no guarantees being made about its efficiency and its effects to the completeness of the search. However, it is an important goal of this work to provide a powerful and efficient set of functions that allow formulation and solution of a wide range of problems. In the view of integrated task and motion planning, the design of this function interface to kinematics and geometry is of crucial importance and subject to on-going research [26, 29]. In the following, we will refer to these domain-specific functions as *geometric predicates*, which appear both in preconditions and effects, and will be discussed in detail in Chapter 4.

In order to discuss our approach to robot task planning, we first describe a task planning scenario that requires only knowledge-level reasoning, the FORCE SENSING scenario. This first scenario is simple enough not to require any geometric predicates and effects—rather than interacting through robotics-specific functions, it suffices to solve task-level actions and later refine these during robot trajectory generation. In further Chapter 6, we will introduce and evaluate more complex scenarios, which require a combined symbolic–geometric solution and demonstrate the full feature set of our KABouM system for integrated task and motion planning.

3.3.3 Force Sensing Scenario

As described in previous work [7], the FORCE SENSING scenario contains a compliant robot manipulator that is supposed to transfer n beverage containers from one support surface to another. Figure 3.1 shows an overview of a setup with two containers, which was planned and demonstrated with the seven degrees-of-freedom LBR4 robot. In this problem scenario, a container may be filled with a liquid, in which case the robot must transport it upright in order to avoid spilling any liquid.



Figure 3.1: Implementation of a FORCE SENSING instance with two objects, a torque-sensing compliant manipulator and a force-controlled parallel gripper [7, 9]. The LBR4 robot is supposed to transfer beverage containers to another table, and it must hold containers upright to prevent spilling unless they are known to be empty.

Containers that are completely empty can safely be transferred on arbitrary trajectories, and the robot should choose a shorter trajectory in such case. In order to determine whether a container is empty or can potentially be spilled, the robot can measure its weight by force sensing. In the experiment setup, the residual force in the tool space of the robot is reported from a joint impedance controller that relies on internal torque sensors. Measurements are sufficiently accurate when the robot keeps a static, non-degenerate position. Of course, only a grasped object can be weighed.

Force Sensing Domain

The symbolic actions of the domain definition are listed in Table 3.1. There are two variants of transfer actions available, a `transferUpright` and a `transferFast` action. To apply a transfer action to an object o , the robot must have grasped that object, $K(\text{isGrasped}(o))$, and it must know the truth value of whether $\text{isSpillable}(o)$ to decide which transfer action is allowed. The knowledge of *whether* the object o can be spilled becomes known when the action `senseWeight(o)` is executed in the real world.

As a planning-time effect, the formula $\text{isSpillable}(o)$ will be added to the knowledge database \mathcal{K}_w , contrary to regular fluent knowledge.

Table 3.1: Symbolic action definition \mathcal{A} of the FORCE SENSING scenario.

Action	Preconditions	Effects
$\text{senseWeight}(o)$	$K(\text{isGrasped}(o))$ $\neg K_w(\text{isSpillable}(o))$	$\text{add}(\mathcal{K}_w, \text{isSpillable}(o))$
$\text{transferFast}(o)$	$K(\text{isGrasped}(o))$ $K(\neg \text{isSpillable}(o))$ $K(\neg \text{isRemoved}(o))$	$\text{add}(\mathcal{K}_f, \text{isRemoved}(o))$
$\text{transferUpright}(o)$	$K(\text{isGrasped}(o))$ $K(\text{isSpillable}(o))$ $K(\neg \text{isRemoved}(o))$	$\text{add}(\mathcal{K}_f, \text{isRemoved}(o))$
$\text{grasp}(o)$	$K(\text{emptyGripper})$ $K(\neg \text{isRemoved}(o))$	$\text{add}(\mathcal{K}_f, \text{isGrasped}(o))$ $\text{add}(\mathcal{K}_f, \neg \text{emptyGripper})$
$\text{ungrasp}(o)$	$K(\text{isGrasped}(o))$ $K(\text{isRemoved}(o))$	$\text{add}(\mathcal{K}_f, \neg \text{isGrasped}(o))$ $\text{add}(\mathcal{K}_f, \text{emptyGripper})$

In order to produce correct sequences of grasping and transfer actions, the state is managed by a nullary predicate emptyGripper and a unary predicate $\text{isRemoved}(o)$. A small number of preconditions in the action definitions are optional and are added only for efficiency: the condition that the $\text{isSpillable}(o)$ predicate should not be known before measuring and the queries to isRemoved in the grasp and ungrasp actions.

To complete the symbolic domain definition $\Sigma = (\mathcal{S}, \mathcal{A}, I, G)$, goal criteria and an initial setting of the knowledge databases need to be given. In the case of a scenario instance with two objects o_1 and o_2 , the initial state I and goal definition G would be defined as

$$I = K(\text{emptyGripper}) \wedge K(\neg \text{isRemoved}(o_1)) \wedge K(\neg \text{isRemoved}(o_2))$$

$$G = K(\text{emptyGripper}) \wedge K(\text{isRemoved}(o_1)) \wedge K(\text{isRemoved}(o_2)) .$$

PKS provides a few syntactical extensions to express all-quantifiers and arrays of symbols, such that repetitive symbols o_1, o_2, \dots need not be explicitly enumerated in larger instances.

Solution of the Force Sensing Scenario

Since this scenario contains predicates whose values are resolved at run-time, its solution is a plan with branches rather than a sequence. Precisely, the generated

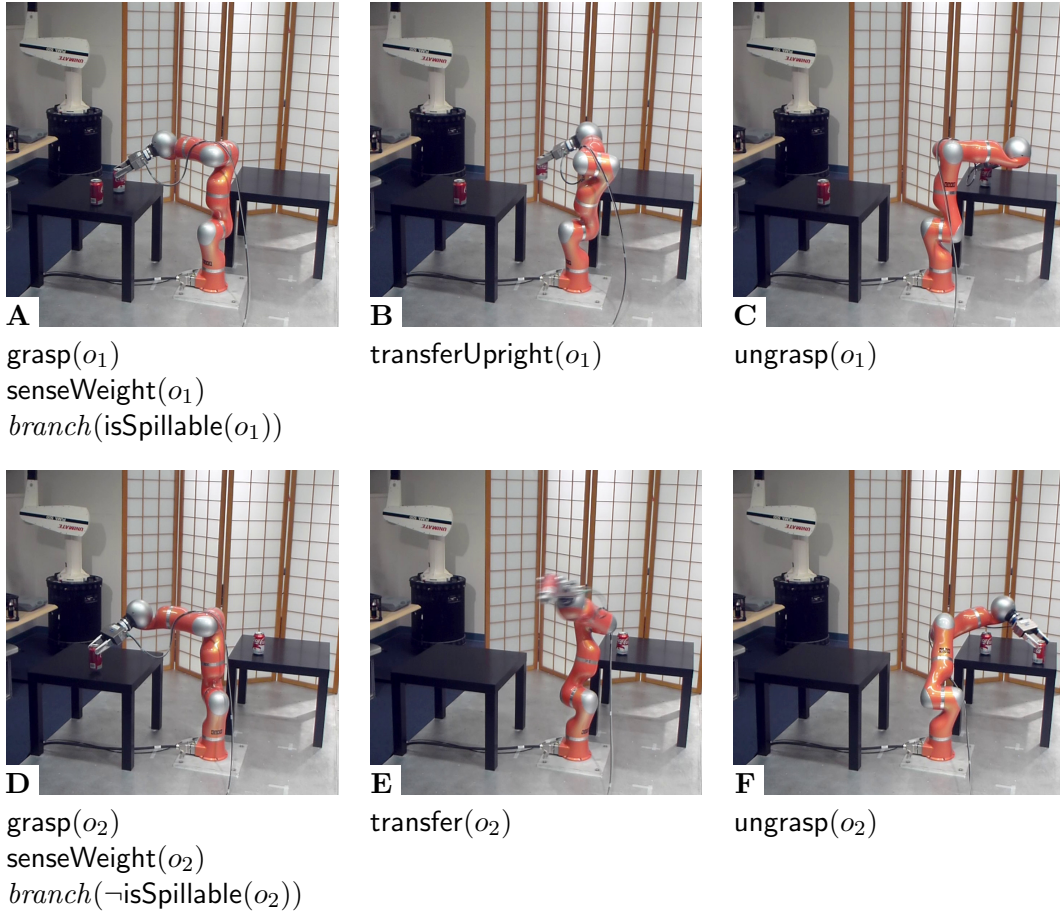


Figure 3.2: Solution of a FORCE SENSING instance with two objects [7, 9]. The LBR4 robot grasps objects and weighs them at run-time (images **A** and **D**), and then follows the correct branch depending on these measurements. Only containers that are known to be empty may be transferred on a fast, direct path (image **E**). Otherwise, the object must be transferred in an upright pose (image **B**).

plan is a tree of actions with sets of symbols as parameters. Each junction defines a ground atomic formula that will be resolved from the preceding sensing action, with a positive and a negative branch to follow depending on its truth. In order to execute this branched plan, an execution component starts with the action at the root of the plan. The implementation of each type of action refines the symbolic action instance and controls the robot and its sensors. In this scenario, refinement of the manipulation actions involves solving inverse kinematics in a simple, posture-optimizing grasp scheme, generating paths with task space constraints, and smooth trajectory interpolation. This first scenario is kept intentionally simple with respect to collisions, which are effectively avoided by lifting objects high enough over the table area when they are transferred. For a sensing action, its action implementa-

tion interprets measurements and generates a list of resolved binary values. The execution component then follows the appropriate branch in the plan.

Figure 3.2 shows a solution that was executed on the real robot setup. The robot grasps the first object and lifts it slightly in order to sense its weight by measuring internal torques and calculating external forces to the tool frame (Figure 3.2 **A**). In this example, sensor measurements indicate that container o_1 could be spilled; task execution follows the appropriate branch in the plan and generates a trajectory that locks both tilt axes in the robot tool frame (Figure 3.2 **B**). Since the second container o_2 is measured to be empty, it is transferred on a faster, unconstrained trajectory (Figure 3.2 **D–F**).

Discussion

On the task planning level, the FORCE SENSING scenario shows an inter-dependence between sensing actions and manipulation actions [7]. The manipulation actions `transferUpright` and `transferFast` require the result of a sensing action, which in turn requires previous application of the manipulation action `grasp`. One may hypothesize that the inter-dependence between sensing and acting is a common property of real-world robot tasks. Importantly, the gain of information by this sensing action is directly formulated in the modal language of knowledge. In this knowledge-level formalism, a sensing action does not need to be formulated as a task of its own, but is implicitly planned in order to gain knowledge required by subsequent actions. Clearly, knowledge-level planning is well suited to generate plans in scenarios with discrete uncertainty and actions that result in gain or loss of information.

3.3.4 Conclusion

In the beginning of this chapter, we have defined the integrated task and motion planning problem. In addition to the symbolic domain definition Σ , the integrated problem also defines robot kinematics and geometric models, and requires geometric preconditions to be fulfilled and geometric effects to be applied. We then discussed the state of the art in automated planning, and chose Planning with Knowledge and Sensing (PKS) as a symbolic planner. PKS plans on the knowledge level, can reason under discrete uncertainty, and can generate plans with branches. We demonstrated its contingency planning behavior with the FORCE SENSING scenario, where a robot needs to transfer objects depending on run-time force measurements, which leads to branches in the plan.

We also discussed the symbolic interface to robotics-specific functions. The FORCE SENSING scenario is a separable problem, where we can first solve the sym-

bolic problem, and then refine actions in the symbolic solution with collision-free paths. It allows a two-step search of symbolic and geometric planning, very similar to the Shakey system [30]. However, only the most simple task and motion planning problems can be separated, while more interesting, real-world problems can only be solved in a combined search space [3]. In order to combine task and motion planning, we define a mapping through geometric predicates for queries from the symbolic to the geometric level, and constraint space sampling to instantiate geometric states that fulfill certain symbolic conditions.

In the following chapter, we define a set of geometric predicates to abstract from geometric states to symbolic predicates. This discussion leads us to the derivation of single-sided approximate collision and inclusion queries, which are vital to almost all robotics scenarios. After discussing the efficient evaluation of these predicates, we complete our symbolic–geometric mapping by describing a method for sampling with geometric constraints, which is followed by the evaluation of several integrated task and motion planning problems on the KABouM system.

Chapter 4

Bounding Meshes for Efficient Geometric Predicates

Efficient geometric data structures and algorithms are key to integrated robot task and motion planning. Our Knowledge-level Action and Bounding Geometry Motion planner (KABouM) relies on and reasons on the level of geometric predicates, in particular collision and inclusion among all geometric entities—objects, robots, and swept volumes of robot motions. Therefore, the efficiency of these geometric queries has a high impact on planning performance and the complexity of the problems that can be solved. In this chapter, we will derive a new algorithm, the bounding mesh algorithm, to generate bounded approximations of geometric entities. This new geometric representation can increase the performance of all geometric queries of the planner. In contrast to earlier approaches, the geometric approximation is strictly single-sided, which is a favorable property in the robotics domain. In short, the task and motion planner detects all collisions and detects them fast, it may only overlook narrow passages thinner than a parameter ε . Essentially, it takes advantage of the asymmetric tolerances of collision and inclusion queries.

The central idea of this chapter is to elaborate the notion of *bounded geometric predicates*, and to derive algorithms to evaluate these queries efficiently. First, we give a mathematical definition for the single-sided approximation and boundedness of this type of predicates. Then, we derive the bounding mesh algorithm as a single-sided approximation of arbitrary geometric shapes. The discussion of bounding meshes is accompanied by a more general evaluation of potential applications to a wider range of problems in collision checking and motion planning. Finally, we complete our discussion on geometric predicates by presenting efficient algorithms for their evaluation and by giving notes on their implementation in the KABouM framework.

4.1 Bounded Geometric Predicates

Robot task and motion planning lends itself to single-sided approximation of collision and inclusion predicates. As mentioned earlier, typical robot task and motion planning problems strictly avoid collisions, because collisions may cause damage to humans, robots, or objects. Even in less severe cases, collisions typically have unforeseen effects in the physical world and may render the task infeasible. Reporting a collision in a borderline case is therefore a reasonable design choice for a single-sided approximation. Inclusion queries appear less frequently in task planning scenarios—for instance as part of goal regions for object placement or for defining container objects, and they behave in a way opposite to collision queries. Ignoring a geometric inclusion will not produce incorrect plans in typical scenarios; it will

at most affect the search space. Of course, we would like to control the precision of this approximation in a reasonable way, because heuristics too arbitrary or too coarse can easily affect the quality of the task and motion planner. A meaningful way to control the level of approximation of collision and inclusion queries is to guarantee a maximum geometric distance ε from the exact query, which we specify in the following. Contrary to many earlier approaches to mesh approximation, it is important to note that we apply only a single-sided approximation. Collisions may be reported for distances less than ε , and inclusions may be ignored for penetration depths less than ε . In all other cases, including all collisions and non-inclusions, the predicates must be exact.

In order to provide a formal definition of the bounded geometric predicates, we cover all four cases of geometric queries that we implement in the KABouM system, including collisions with swept volumes of robot motions. Let m_0, m_1 denote geometric models whose border is given by closed triangle meshes, and Q_0, Q_1 denote continuous paths in an Euclidean space. r_0, r_1 refer to robot kinematics whose definition includes a continuous mapping of these paths to $SE(3)$. For a robot r_0 and a path Q_0 within the configuration space of its kinematic, $SV(r_0, Q_0)$ defines the volume swept by its geometry along that path. For brevity, our definition uses signed distances, where negative values represent penetration depths.

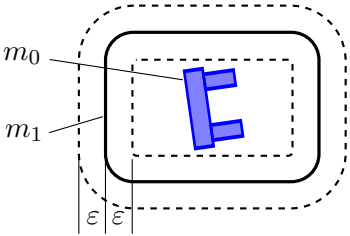
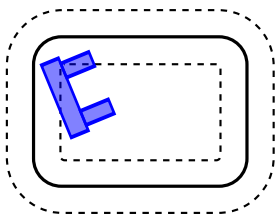
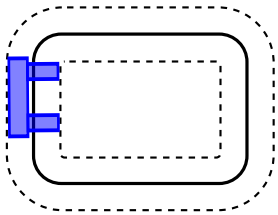
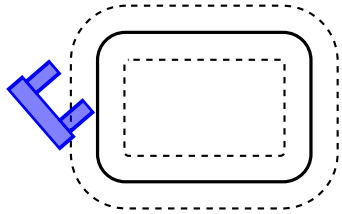
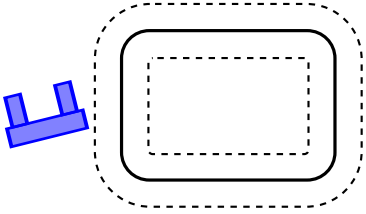
Definition 1 (Bounded Geometric Predicates). *We define the following binary geometric predicates for collision and inclusion.*

1. $m_0 \cap m_1 \neq \emptyset$, whether two geometric models collide.
2. $SV(r_0, Q_0 = [q_0, q_1, \dots]) \cap m_1 \neq \emptyset$, whether the swept volume of a robot motion collides with an object.
3. $SV(r_0, Q_0) \cap SV(r_1, Q_1) \neq \emptyset$, whether the swept volumes of two robot motions collide.
4. $m_0 \subseteq m_1$, whether a geometric model includes another.

Their evaluation may use a single-sided ε -precise approximation: Collision predicates (1–3) may return arbitrary answers for signed distances within $[0, \varepsilon]$, and the inclusion predicate (4) may return arbitrary answers for signed distances within $[-\varepsilon, 0]$; otherwise, their answer is exact.

Table 4.1 visualizes our definition of bounded geometric predicates in a truth table.

Table 4.1: Definition and truth table of the bounded collision and inclusion predicates in terms of signed distances. The values of the corresponding exact predicates are given in brackets, where they differ. Note that in the inequalities that define each case, d refers to the range of signed distance values over all points of m_0 , in contrast to the shortest distance alone.

Signed distances d to m_1 , measured from all points of m_0	Visual Example	Bounded Collision Predicate $m_0 \cap m_1 \neq \emptyset$	Bounded Inclusion Predicate $m_0 \subseteq m_1$
$d < -\varepsilon$		true	true
$d < 0$		true	arbitrary (ideally, true)
$-\varepsilon < d < \varepsilon$		true	false
$0 < d$		arbitrary (ideally, false)	false
$\varepsilon < d$		false	false

Design of the Bounded Geometric Predicates

Our definition of bounded geometric predicates with their single-sided, ε -precise collision and inclusion queries is chosen with two thoughts in mind. The first reason is that in robot task and motion planning, it is *sufficient* to provide geometric predicates that are exact on one side of the approximation. This property of collision and inclusion queries was already explained in earlier discussions. The second reason is that approximation is *necessary*. Geometric queries are the computational bottleneck in most task and motion planners, including ours, and exact queries are computationally expensive. In order to increase the possible size and complexity of problems that the system can solve, an approximation is therefore necessary. Based on these two reasons, we developed the bounding mesh and bounding sets of convex polyhedra approximations of geometric meshes. In the following sections, we derive techniques to generate these structures and apply them to evaluate the bounded geometric predicates at great efficiency.

4.2 Bounding Meshes

Single-sided approximation can both reduce the complexity of geometric shapes and preserve the correctness of geometric queries. In our case, we apply this approximation to implement a set of bounded geometric predicates for integrated task and motion planning. This section is concerned with the general problem of single-sided approximation of triangular meshes. In particular, we develop a new algorithm to generate such a single-sided approximate mesh, which we refer to as a *bounding mesh*. In our definition, a bounding mesh is a mesh that includes an original mesh, and has fewer vertices than the original. Conversely, an *inner bounding mesh* is included by the original mesh. A complete definition of both (outer) bounding meshes and inner bounding meshes is given in the following.

Definition 2 (Bounding Mesh). *For a given mesh M , we may generate a single-sided approximation M' with the following properties.*

- 1a. *If $M' \supseteq M$, M' is a bounding mesh (or, outer bounding mesh) of M .*
- 1b. *If $M' \subseteq M$, M' is an inner bounding mesh of M .*
2. *M' is a simplification of M , it should have fewer vertices.*

We may refer to M' as ε -precise if the Hausdorff distance $d(M, M')$ is no larger than a given parameter ε .

The Hausdorff distance is a commonly used metric to measure the difference between two meshes [72]; it is defined as

$$d(M, M') = \max \left\{ \sup_{p \in M} \inf_{q \in M'} \|p - q\|^2, \sup_{q \in M'} \inf_{p \in M} \|p - q\|^2 \right\}. \quad (4.1)$$

Intuitively, the Hausdorff distance between two meshes is defined as the maximum of the shortest distances to the other mesh over all points on both meshes [12]. Usually, a bounding mesh is supposed to be no further than a distance ε from the original; with this property, it allows a wide range of bounded-approximation algorithms to run efficiently, including our bounded geometric predicates.

4.2.1 Level-of-Detail Models

Of course, mesh approximation and level-of-detail models are well known concepts in computer geometry and related fields. Many geometric algorithms can operate more efficiently when simplified versions of a given mesh are available. As an example, many collision detection routines generate simplified models that enclose the original mesh and operate on bounding volumes in their broad-phase search. Common bounding volume hierarchies include axis-aligned boxes (Figure 4.1b), spheres (Figure 4.1a), oriented boxes, discrete orientation polytopes, and convex hulls (Figure 4.1c) [73, pp. 75ff]. Figure 4.1 shows an overview of the most common types of bounding volumes of a high-resolution mesh, with bounding meshes at different precisions and the original mesh given for comparison. We would like to emphasize that bounding meshes and the broader types of volume hierarchies guarantee to enclose the original mesh, while most other mesh approximations, especially those designed for computer graphics and animation, do not. General, unconstrained mesh simplification was an active field of research in the 1990s, with a wide range of local and global optimization algorithms being developed. Luebke gives an extensive summary on the field of unconstrained mesh simplification in an article [74] and in a book [75]. Cignoni, Montani, and Scopigno also categorize and evaluate existing implementations in a survey article [76].

Sphere Tree Bounding Volumes

The sphere is the type of shape that allows the fastest collision checking. Sphere trees are unions of spheres and can be generated to approximate general shapes. Some algorithms can generate sphere trees that are guaranteed to enclose meshes

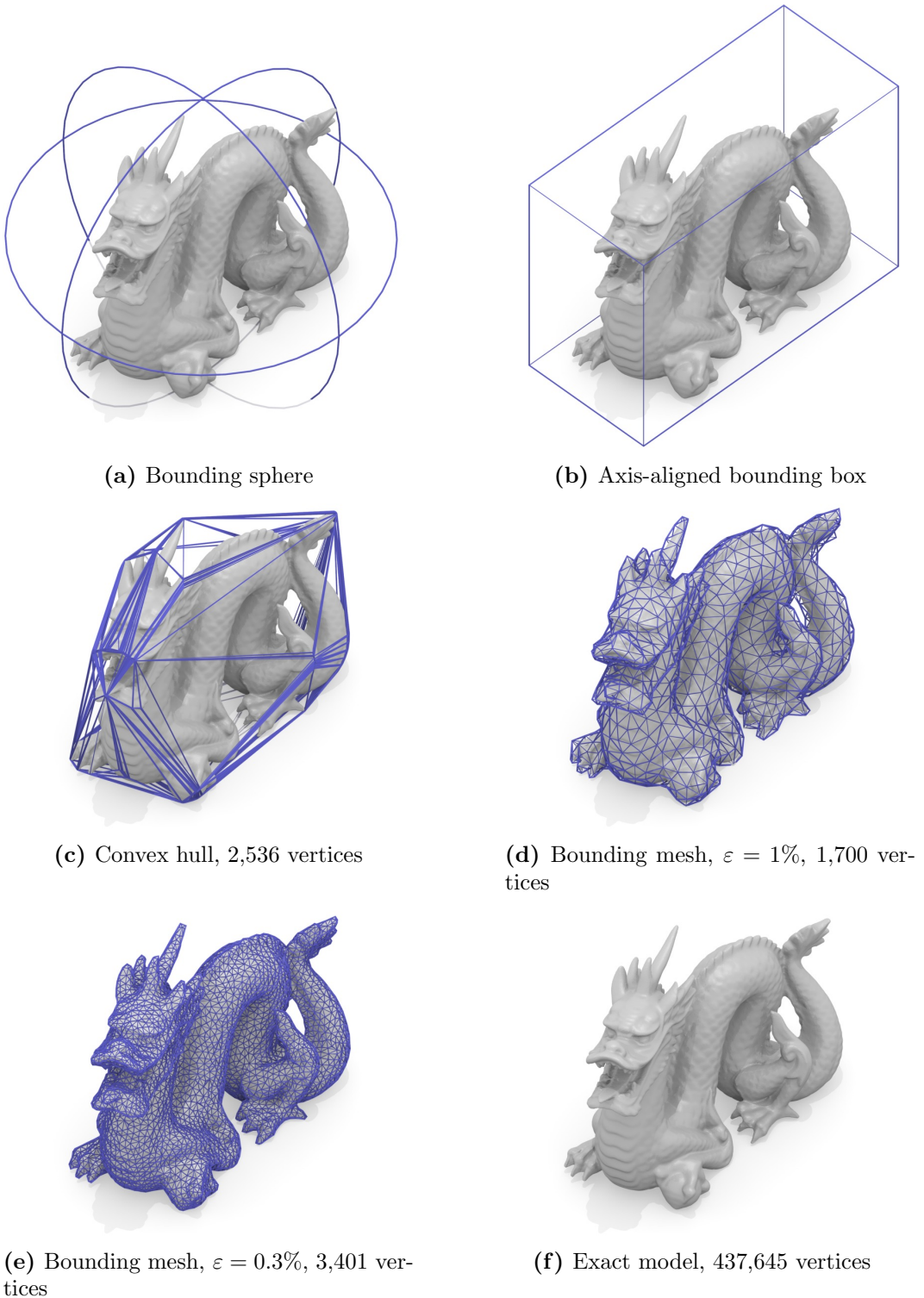


Figure 4.1: Bounding volume hierarchy of a high-resolution mesh. Bounding spheres and axis-aligned boxes are implicitly used in many geometric algorithms; convex hulls are sometimes generated in a pre-processing step. Bounding meshes can be understood as more fine-grained approximations in this hierarchy. The original model was scanned and made available by the Stanford Computer Graphics Laboratory.

(up to a certain sampling resolution) and can therefore serve as bounding volumes, for instance Bradshaw’s adaptive medial-axis approximation [77].

Compared to bounding meshes, sphere trees are much harder to fit tightly within a given tolerance and rather geared towards rough approximation. While sphere trees are suitable for coarse approximation and very fast collision checking, they are ill-suited for fine approximations of flat or concave surfaces, where sphere trees may even require more spheres than the number of triangles of the input mesh [78, p. 139].

Convex Surface Decomposition

Bounding volumes are not the only technique to leverage collision and distance queries. Ehrmann and Lin perform convex surface decomposition and build up a hierarchical data structure to speed up collision detection and other types of geometric queries, packaged in the SWIFT++ collision detection library [79]. It is important to note that surface decomposition is different from regular, convex decomposition of solids. While their exact convex surface decomposition is specifically integrated into the SWIFT++ library, our bounding sets of convex polyhedra generation (Section 4.4) is a pre-processing step independent from and an orthogonal optimization to the actual collision detection library in use, suitable for all bounded ε -precise queries.

While some concepts of mesh simplification can be adapted to bounding approximations—our bounding mesh approach uses a cost metric for decimation operations similar to Garland [80]—we focus our discussion of related works on the much narrower field of bounded mesh approximation.

4.2.2 Single-Sided Mesh Approximation

The idea of generating approximate meshes that enclose a more detailed mesh was probably first outlined in a 1999 patent description by Hoppe [81]. Rather than a single bounding mesh, Hoppe proposes a progressive hull structure described by a sequence of edge contractions, which is designed for progressive transmission and level-of-detail rendering. However, the edge contractions in Hoppe’s progressive hulls are chosen to minimize the added volume, rather than a metric closer to the maximum distance or Hausdorff distance from the simplified mesh to the original. While the added volume metric is easy to compute and minimize, it effectively allows sharp spikes arbitrarily far away from the original model, because spikes add very little volume. The added volume metric may be a reasonable choice for creating

bounding volume hierarchies as a heuristic for graphics algorithms, it is however not suitable for providing distance limits on the simplified mesh or ε -precision, as required for the queries of the KABouM planner.

One such practical application of added volume progressive hulls is that of silhouette clipping. In a 1999 technical report, Gu et al. [82, pp. 21–23] generate a low-resolution bounding mesh that allows fast normal mapping, and then clip its silhouettes to achieve accurate rendering results with this approximate geometry. The silhouette clipping approach was popularized by Sander et al. in 2000 [83]. If detailed, high-quality polygonal models were approximated by low-resolution models and normal mapping, their silhouette would appear very coarse. Sander et al. operate on a bounding mesh for texture and normal mapping, and then clip silhouettes at full resolution using a precomputed edge hierarchy. Essentially, their use of bounding meshes is to generate a rendered view at great efficiency. The rendered view is guaranteed to enclose the original model and can therefore be masked appropriately to achieve a high-quality silhouette.

Platis and Theoharis apply Sander’s added volume progressive hulls to ray intersection point queries with high-resolution meshes [84]. Their method is designed for ray tracing and collision detection tasks with detailed, non-convex meshes. It is further elaborated in Platis’ thesis [85] (in Greek). Compared to Sander, they propose a simpler edge contraction metric to avoid recalculation of all neighboring edges after a contraction, effectively speeding up progressive hull generation. In addition, they add two simple checks to avoid generating sharp angle edges or degenerate triangles. A few technical improvements on hard edges were later proposed by Cholt [86]. The main contribution of Platis and Theoharis is to localize possible areas for ray intersection points faster than with traditional k-DOPs (discrete oriented polytopes) using progressive hulls. While their type of added volume progressive hulls may contain spikes and are not bounded error approximations, this affects only the heuristic of the search for ray intersections, and exact intersection points are computed after selective refinement. In a diploma thesis, Ciesla [87] provides a purely geometric edge contraction scheme that does not optimize a specific cost function.

To the author’s knowledge, the above publications [81, 82, 83, 84, 85, 86, 87] and the author’s works [12, 11] are the only prior works in single-sided mesh simplification, according to an extensive search for publications related to bounding meshes, progressive hulls, and other search terms.

Mesh Simplification within Error Limits

Related concepts were derived in order to guarantee tight distance limits for simplified meshes. An early approach to ε -precise approximation of meshes is covered in a thesis by Varshney [88]. Cohen et al. [89] then develop this further and describe the generation of simplification envelopes, which are similar to offset meshes in both directions. Therefore, simplification envelopes can work in conjunction with most mesh simplification techniques to guarantee an approximation within a bounded distance. Guézic [90] proposes to use tolerance volumes to simplify meshes within distance boundaries. Contrary to our bounding meshes, these approximations are not single-sided, but only ε -precise.

4.3 Bounding Mesh Generation

In the following, we derive an algorithm to generate bounding mesh approximations. While bounding meshes are in no way limited to applications in bounded collision and inclusion checking, our intention is to implement the bounded geometric predicates defined in the beginning of this chapter (Section 4.1). For this reason, the design goals for the bounding mesh generation are as follows:

- The mesh approximation must be single-sided, in other words, we generate inner and outer bounding meshes.
- The distance to the original mesh must be measured such that it can be bounded to a given limit ε .
- The approximation should seek to keep the distance to the original mesh at a minimum. Ideally, the bounding mesh approximation is controlled by the Hausdorff distance to the original or an approximate function thereof.

It turns out that the first condition is comparably simple to achieve in an algorithm that decimates the mesh iteratively. Under the reasonable assumption that the given mesh is a closed, orientable 2-manifold (rather than an arbitrary set of triangles), we only need to show that the iteration step creates a bounding mesh that encloses the previous mesh. Conversely, for creating an inner bounding mesh, we only need to show that each iteration step yields an inner bounding mesh included by the previous mesh. For inner bounding meshes, we additionally need to ensure that self-intersections are either avoided altogether, or at least handled properly in further processing steps (such as convex decomposition). For a single-sided mesh

approximation, it is therefore a reasonable choice to devise an iterative mesh decimation technique. Most mesh decimation algorithms are iterative [74, 76], and in a number of them, the decimation step can be adapted to create a bounding mesh. For those that perform contractions, it is a necessary condition that the contracted region is a single-sided approximation. This enforces constraints on the placement of contracted vertices or edges, and discards simple schemes such as mid-point edge contraction from the list of suitable decimation operators.

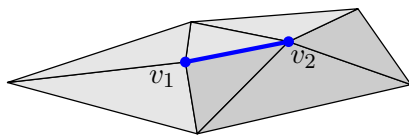
Considering the second condition of the above design goals, a distance limit ε on the approximation can be achieved in two ways. The simplest way to realize distance limits is to perform checks after each step in an iterative algorithm, which is of course computationally expensive. In our further discussion, we will describe a cost function that is an upper bound to the distance function and works as a more direct distance limit (Section 4.3.5).

From our above list, the third design goal, the distance minimization to the original mesh, is the hardest to achieve. While the Hausdorff distance is a good measure for the quality of a mesh approximation, and sufficient to realize ε -precise bounded geometric predicates, it is a complex, non-smooth function unsuitable for direct cost function optimization [72, p. 23]. It is clear that the Hausdorff distance can only be approximated; one such approximation is the quadric error metric, as defined by Garland and Heckbert [80]. To achieve this, we first derive a bounding mesh decimation operation in terms of a generic cost function. Then, we introduce the quadric error metric as a cost function and discuss its optimization specifically.

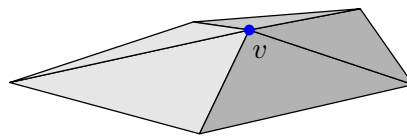
4.3.1 Bounding Mesh Edge Contraction

Following the discussion on the design goals for a single-sided and distance-limited mesh approximation, we design our bounding mesh algorithm based on iterative edge contraction, as summarized in our technical report [12]. An edge contraction is essentially the replacement of an edge e of two neighboring vertices (v_1, v_2) by a single vertex v , with the neighboring triangles being adjusted to this new geometry. For a bounding mesh edge contraction, the new neighborhood must enclose the older one. To achieve this, it is sufficient to restrict v to be above the planes of the neighboring triangles. Note that in bounding mesh approximation, v cannot generally be the midpoint of (v_1, v_2) in order to meet this criterion, as opposed to simpler contraction schemes.

A rather important choice in our bounding mesh algorithm is the design of the edge contraction cost function $E : (e, \mathbf{v}) \mapsto \mathbb{R}$, where e is an edge of points $(\mathbf{v}_1, \mathbf{v}_2)$ that are contracted to a single vertex \mathbf{v} , as illustrated in Figure 4.2 [12]. The goal



(a) Edge (v_1, v_2) with neighboring triangles



(b) Local bounding mesh with new vertex v

Figure 4.2: Bounding mesh edge contraction.

of the bounding mesh edge contraction is to find a contraction point \mathbf{v}^* that is guaranteed to be outside the mesh and has minimal cost $E(e, \mathbf{v}^*)$. Denoting P as the neighboring planes of a vertex, we can formulate the search for the contraction point \mathbf{v}^* as a constrained minimization problem

$$\operatorname{argmin}_{\mathbf{v}^*} E((v_1, v_2), \mathbf{v}^*) \quad \text{s. t.} \quad \forall \mathbf{p} \in P(v_1) \cup P(v_2) : \mathbf{p}^T \mathbf{v}^* \geq 0. \quad (4.2)$$

Contrary to all earlier approaches to bounding mesh generation [81, 82, 83, 84, 86], we do *not* formulate E in terms of the volume added by the edge contraction. The added volume is a sum of tetrahedra placed on $P(v_1) \cup P(v_2)$, and therefore a function linear in \mathbf{v} , which can be solved by linear optimization [82, 84]. While some computer graphics applications do not require approximations at close distances and can be leveraged by volume-minimized bounding meshes [83, 84], added volume is too weak of an estimate for the Hausdorff distance. Most importantly, the added volume may become infinitely small in common cases such as sharp angles or thin triangles, effectively allowing \mathbf{v} to move infinitely far away from the original mesh and generating spikes in the bounding mesh. For this reason, volume-minimized bounding meshes are not suitable for applications with distance limits, including our bounded geometric predicates for robot task and motion planning.

4.3.2 Quadric Error Metric

The quadric error metric closely reflects the Hausdorff distance of an edge contraction [72, p. 79f.], especially in comparison to the added volume cost function used in earlier approaches to bounding mesh generation [81, 82, 83, 84, 86].

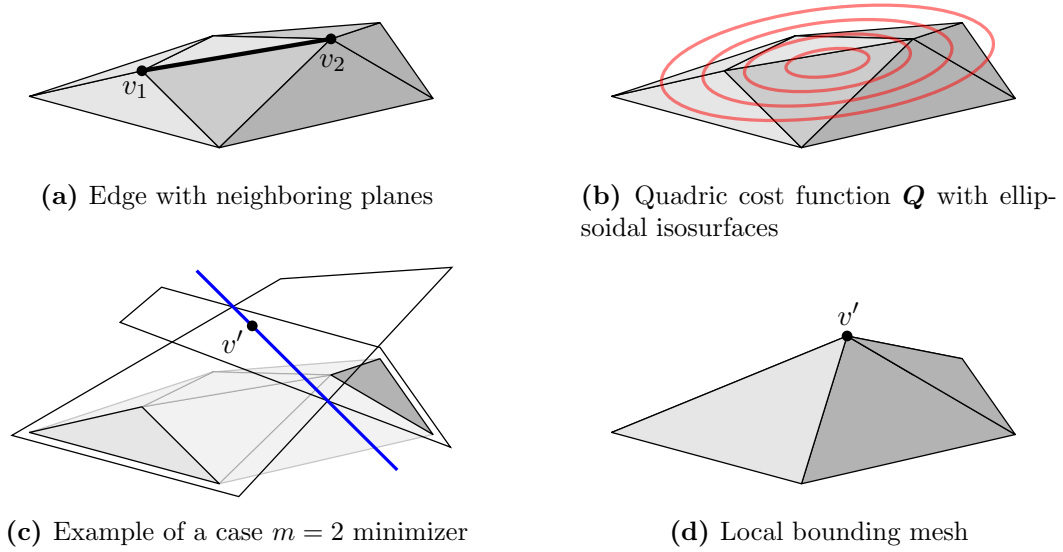


Figure 4.3: Bounding mesh edge contraction with a quadric cost function. [12]

In its most general form, a quadric Q maps a 3D coordinate $\mathbf{v} \in \mathbb{R}^3$ to a squared distance measure $d^2 \in \mathbb{R} \geq 0$.

$$Q : \mathbf{v} \mapsto \begin{bmatrix} \mathbf{v}^T & 1 \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{v} \\ 1 \end{bmatrix} \quad (4.3)$$

A more mathematically complete definition for our notation of quadric functions is given in Appendix A.1, together with their basic properties, operations, and distance measures to basic geometric shapes.

The general approach of edge contraction with quadric costs is to approximate the Hausdorff distance of the contraction by the simpler sum of distances from the new vertex v to the neighboring planes $P(e)$ of the contracted edge. Let d denote the distance of two geometric shapes. Then, the quadric cost of the edge contraction can conveniently be written as a multiplication with a symmetric 4-by-4 matrix, the quadric $\mathbf{Q}(e)$.

$$E(e, \mathbf{v}) = \sum_{\mathbf{p} \in P(e)} (d(\mathbf{p}, \mathbf{v}))^2 = \mathbf{v}^T \sum_{\mathbf{p} \in P(e)} \mathbf{p}\mathbf{p}^T \mathbf{v} = \mathbf{v}^T \mathbf{Q}(e) \mathbf{v} \quad (4.4)$$

Figure 4.3 visualizes the behavior of this type of cost function. A quadric possesses (possibly degenerate) ellipsoidal isosurfaces, as shown in Figure 4.3b.

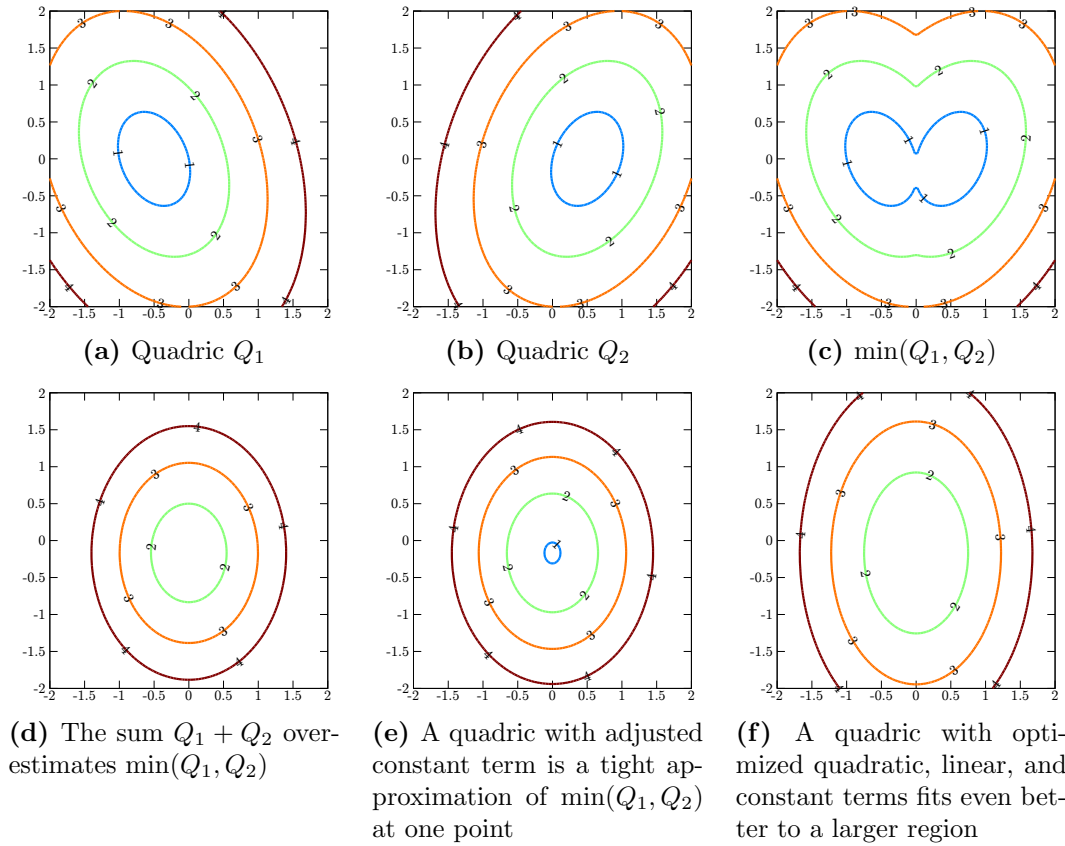


Figure 4.4: The sum of two quadrics generally overestimates their minimum. Tighter approximations of $\min(Q_1, Q_2)$ can be achieved by adjusting the constant term or all coefficients of the quadric.

4.3.3 Quadric Cost for Compound Shapes

It is a rather important property of the cost function how it is affected by contracted and newly created edges. We cannot directly re-calculate Eq. 4.4 for modified edges, as this would ignore the distance to the original mesh, and allow it to move further away in each iteration. The most commonly proposed update operation for quadric costs on the decimated mesh is the addition of the quadrics, which is effectively an upper estimate of the original sum of distances to all neighboring planes.

$$Q(\mathbf{v}') = Q(\mathbf{v}_1) + Q(\mathbf{v}_2). \quad (4.5)$$

This approximation greatly overestimates the Hausdorff distance after several neighboring edges have been contracted; it may even overestimate Eq. 4.4 by a factor of three [80] and potentially causes suboptimal bounding meshes as a result.

The underlying issue is that a quadric can give the exact squared distance only for the primitive shapes points, lines, or planes. In contrast, distances to more general compound shapes can only be approximated, let alone the Hausdorff distance between two meshes. In general, this problem can be reduced to approximating a compound of two quadrics Q_1 and Q_2 , whose distance is $\min(Q_1, Q_2)$, by an upper approximation $Q \succeq \min(Q_1, Q_2)$. (For details on the relation \succeq , see Appendix A.1.)

An example of two simple quadrics is shown in Figure 4.4a. The distance to their compound shape (Figure 4.4c) is only piecewise quadratic, and quadrics can only approximate this type of function. The simple sum $Q_1 + Q_2$ would be a weak upper bound (Figure 4.4d), distances to triangle meshes or compounds of primitive shapes would be largely overestimated, especially after several additions.

Approximation with Adjustment of the Constant

A slightly tighter approximation is possible with an adjustment of the constant term of the quadric. Starting with the simple sum $Q = Q_1 + Q_2$, we can reduce its constant term $Q_{4,4}$ such that it contacts $\min(Q_1, Q_2)$ in at least one point, for which we choose the minimizer $\mathbf{m} = \operatorname{argmin}(Q)$. (A closed-form solution for this minimum operator is given in Appendix A.1.2.) Now, we can safely subtract $\max(Q_1(\mathbf{m}), Q_2(\mathbf{m}))$ from $Q_{4,4}$ in order to achieve the contact point $Q(\mathbf{m}) = \min(Q_1(\mathbf{m}), Q_2(\mathbf{m}))$. Since the curvature of Q is greater or equal than that of Q_1 and Q_2 , it is an upper approximation $Q \succeq \min(Q_1, Q_2)$, as intended. Resuming our discussion above, Figure 4.4e shows an example of this slightly improved approximation in comparison to the simple addition of quadrics.

Approximation with Adjustment of all Coefficients

Since we can minimize the constant of a quadric, an additional step would be to adjust the remaining coefficients of the quadric to achieve more contact points and obtain an approximation that is locally optimal. Obviously, there is no clear, global optimum to fit a quadric to shapes like Figure 4.4c, but we can construct a locally optimal approximation invariant to transformation and symmetry, described in the following.

To achieve invariance to transformations, we first factorize $Q_1 + Q_2$ to $\mathbf{T}^T \mathbf{S} \mathbf{T}$ and obtain a reference frame \mathbf{T} . For the derivation of the factorization procedure, refer to Appendix A.1.2. Our approach is to choose this \mathbf{T} as a position and an orientation for the quadric Q to be constructed. Note that, when Q_1 and Q_2 are sufficiently close and similar, their representations in frame \mathbf{T} , \mathbf{S}_1 and \mathbf{S}_2 , will already be almost diagonal. In order to construct Q , we first bound its quadratic terms and then its

linear terms, taking care of dependencies between the coefficients. First, we use the fact that a symmetric matrix \mathbf{S} can be bounded by a diagonal matrix \mathbf{S}' .

$$\mathbf{x}^T \mathbf{S} \mathbf{x} = \mathbf{x}^T \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix} \mathbf{x} \leq \mathbf{x}^T \begin{bmatrix} \sqrt{s_{11}^2 + s_{12}^2} & 0 \\ 0 & \sqrt{s_{12}^2 + s_{22}^2} \end{bmatrix} \mathbf{x} = \mathbf{x}^T \mathbf{S}' \mathbf{x} \quad (4.6)$$

With this, we can diagonalize the upper left 3-by-3 quadratic parts of \mathbf{S}_1 and \mathbf{S}_2 . Of these diagonalized matrices, we may take a simple maximum to obtain \mathbf{S}' .

In a second step, we ensure that \mathbf{Q} bounds the linear terms of \mathbf{Q}_1 and \mathbf{Q}_2 . We use the fact that a linear term tx can be bounded by a quadratic term $sx^2 + c$.

$$sx^2 + c = \frac{t}{2l}x^2 + \frac{tl}{2} = \frac{t}{2} \left(\frac{x^2}{l} + l \right) = \frac{t}{2} \left(\frac{x}{\sqrt{l}} - \sqrt{l} \right)^2 + tx \geq tx \quad (4.7)$$

Here, $l > 0$ is a free parameter that allows us to choose at which distance the approximation is exact.

Applying these two steps, we can construct a diagonal matrix \mathbf{S}'' that is greater than both $\mathbf{T}^{-T} \mathbf{Q}_1 \mathbf{T}^{-1}$ and $\mathbf{T}^{-T} \mathbf{Q}_2 \mathbf{T}^{-1}$. This also holds for the final result $\mathbf{Q} = \mathbf{T}^T \mathbf{S}'' \mathbf{T}$. By construction, this type of approximation is invariant to transformation and symmetry, and optimizes all coefficients of the quadric. Figure 4.4f shows an example of the result, which fits well to a larger region than just its minimum (Figure 4.4e), controlled by parameter l .

4.3.4 Optimal Edge Contraction

In the previous section, we discussed the choice of the edge contraction cost function E , and we designed a quadratic cost function Q as a feasible approximation of the Hausdorff distance. In this section, we derive an efficient closed-form solution to minimize these quadric costs and to find a minimizing edge contraction point outside the local mesh. Obviously, the problem of finding the optimal edge contraction point \mathbf{v} outside of the local mesh $P(\mathbf{v}_1) \cup P(\mathbf{v}_2)$ and with minimal cost $\mathbf{v}^T \mathbf{Q} \mathbf{v}$ is, in general, a quadratic minimization with linear inequalities:

$$\underset{\mathbf{v}}{\operatorname{argmin}} \mathbf{v}^T \mathbf{Q} \mathbf{v} \quad \text{s. t.} \quad \forall \mathbf{p} \in P(v_1) \cup P(v_2) : \mathbf{p}^T \mathbf{v} \geq 0 \quad (4.8)$$

To derive a stable and efficient solution, we exploit the geometric structure of this problem. Since the unknown \mathbf{v} is only a point in 3D space, it can at most be constrained by three linearly independent inequalities. Because of this, it is not necessary to solve the generalized quadratic problem with potentially many inequalities. We rather notice that there exists a subset of [0..3] inequalities, whose

minimizer equals the minimizer of the general problem. Therefore, we can assume that $m \in [0..3]$ inequalities hold as equalities, and exhaustively search in all subspaces that are given by all these subsets of equalities. More precisely, we enumerate all m -subsets of neighboring planes $P(\mathbf{v}_1) \cup P(\mathbf{v}_2)$. For each subset, we assume its m planes constrain the minimizer, and solve for a candidate minimizer in the linear subspace formed by the m equalities. From all candidate minimizers, we select those that fulfill all other inequalities, of which we pick the cheapest, which is the global minimizer \mathbf{v} . With this approach, we effectively reduce the quadratic problem with inequalities to a set of simpler quadratic problems with only m equalities.

Closed-Form Solution of the Edge Contraction

Edge contraction points can be optimized by solving a set of quadratic optimization problems with up to three equality constraints. In the following, we follow this idea to derive a generic closed-form solution and a more stable specialized solution. Finally, we propose an algorithm to solve the bounding edge contraction problem.

Considering the neighboring planes of the edge $(\mathbf{v}_1, \mathbf{v}_2)$ that is to be contracted, we denote \mathbf{N} as their normals, and \mathbf{d} their signed distances to the origin. With this notation, we would like to find the optimal contraction point \mathbf{v} . In order to ease calculations, we define \mathbf{v} in homogeneous coordinates, i.e. $\mathbf{v} \in \mathbb{R}^3 \times \{1\}$. In this notation, the contraction point solution is then written as follows:

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{Q} \mathbf{v} \quad \text{s. t.} \quad \begin{bmatrix} \mathbf{N} \\ -\mathbf{d} \end{bmatrix}^T \mathbf{v} \geq \mathbf{0} \wedge \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{v} = 1 \quad (4.9)$$

Following our previous discussion, we may assume that the global minimizer \mathbf{v} is constrained by a subset of $m \in [0..3]$ of the above inequalities, and these subset constraints hold as equalities. In the exhaustive search of our algorithm, we can therefore solve for candidate minimizers $\mathbf{x} \in \mathbb{R}^3 \times \{1\}$ with respect to each m -subset, and obtain the global minimizer by comparison. Let m denote the number of equality constraints of a given subset, and $\begin{bmatrix} \mathbf{N}^T & -\mathbf{d}^T \end{bmatrix}$ now denote the equalities of this subproblem. Then, we are to solve

$$\min_{\mathbf{x}} \mathbf{x}^T \mathbf{Q} \mathbf{x} \quad \text{s. t.} \quad \begin{bmatrix} \mathbf{N} \\ -\mathbf{d} \end{bmatrix}^T \mathbf{x} = \mathbf{0} \wedge \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{x} = 1. \quad (4.10)$$

A generic way to solve this subproblem would be to introduce Lagrangian multipliers $\boldsymbol{\lambda} \in \mathbb{R}^{m+1}$ and add the summand $\boldsymbol{\lambda}^T \begin{bmatrix} \mathbf{N}^T & -\mathbf{d}^T & \mathbf{0} \\ \mathbf{0}^T & 1 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ to the optimization function,

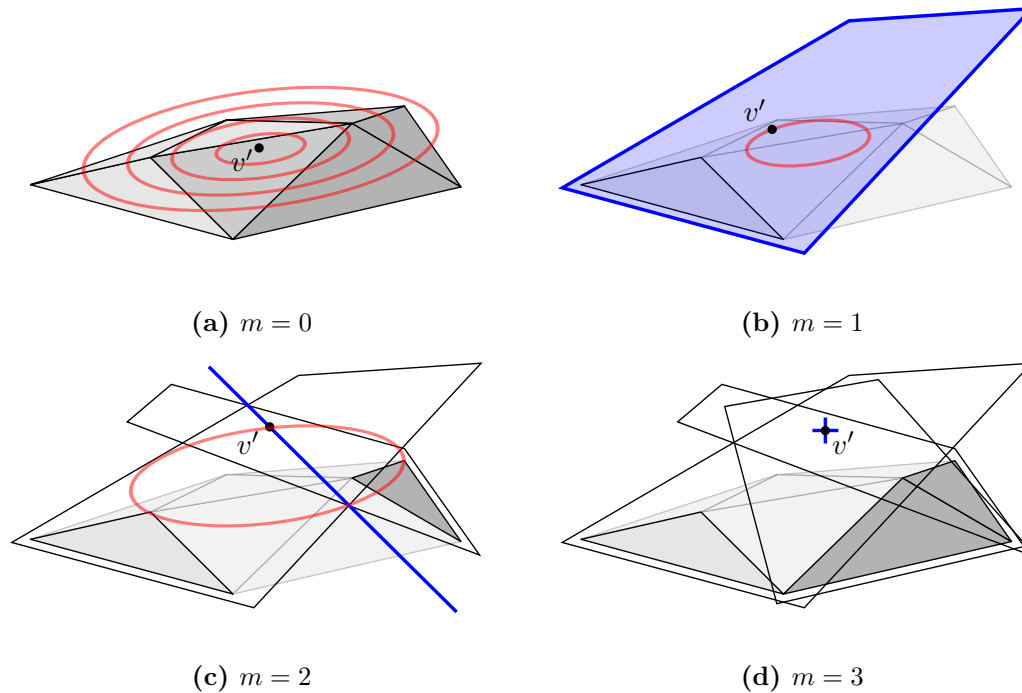


Figure 4.5: An optimal edge contraction point may be bounded by zero, one, two, or (at least) three planes, leading to four different cases in the solution. Intuitively, these constraints describe linear subspaces (shown in blue). In a minimizer v' , the constraint subspace must be tangential to a surface of equal costs (shown in light red).

effectively defining an unconstrained problem. After symbolic differentiation, we can reduce its minimization to solving the following linear problem of dimension $m + 5$:

$$\begin{bmatrix} \mathbf{Q} & \mathbf{N} & \mathbf{0} \\ \mathbf{N}^T & -d & 1 \\ \mathbf{0}^T & 1 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \quad (4.11)$$

However, the Lagrangian solution is rather badly conditioned, especially for $m = 3$, because the neighboring plane normals in \mathbf{N} are often parallel or close to parallel. To overcome these numerical problems, we devise a closed-form solution that explicitly models the constraint subspace and minimizes in that subspace. The direct solution operates on the matrix \mathbf{Q} , which is symmetric and positive definite, and therefore better conditioned.

Direct Subspace Solution of the Edge Contraction

For the direct linear subspace solution, we substitute \mathbf{x} by a vector $\mathbf{y} \in \mathbb{R}^{3-m}$ that spans the constrained subspace:

$$\{\mathbf{x} = \mathbf{A}\mathbf{y} + \mathbf{b} \mid \mathbf{N}^T \mathbf{x} = \mathbf{d}^T\} \quad (4.12)$$

With this substitution, the minimization problem is reduced to an unconstrained one of dimension $3 - m$

$$\min_{\mathbf{y}} \mathbf{y}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{y} + 2 \mathbf{Q} \mathbf{A} \mathbf{b} + \mathbf{b}^T \mathbf{Q} \mathbf{b} \quad (4.13)$$

which is solved by the simple linear system

$$\mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{y} = -\mathbf{A}^T \mathbf{Q} \mathbf{b} . \quad (4.14)$$

Of course, in order to perform the substitution, we need to calculate a valid point \mathbf{b} in the subspace and also its range matrix \mathbf{A} . A generic solution would be to numerically solve for $\mathbf{b} = -\text{pinv}(\mathbf{N}) \mathbf{d}$ and $\mathbf{A} = \text{null}(\mathbf{N})$, where pinv denotes the Moore-Penrose pseudoinverse and null the nullspace. However, it is substantially more reliable and more efficient to exploit the geometric meaning of this substitution and solve all four cases $m \in [0..3]$ symbolically. Figure 4.5 visualizes these four cases with their different types of constraint subspaces. The quadric \mathbf{Q} can be thought of as surfaces of equal costs, generally shaped as ellipsoids. As an example, Figure 4.5a depicts surfaces of equal costs as light red ellipsoids, with the minimizer v' located in the center. Discriminating the four cases of $m = [0..3]$, we can derive a closed-form definition of the linear subspaces $\mathbf{A}\mathbf{y} + \mathbf{b}$.

Case $m = 0$: Even without any constraints, it is still useful to apply the substitution in order to obtain a direct solution in \mathbf{y} without homogeneous coordinates. Such a substitution can be obtained with

$$\mathbf{A} = \mathbf{I}_{4 \times 3}, \mathbf{b} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T . \quad (4.15)$$

Case $m = 1$: When the subspace is a plane, its range may be calculated from the vector product of its normal \mathbf{n}_1 with an arbitrary unit-length axis \mathbf{e} . Implementation should choose from two axes, favoring the more linearly independent axis of the two with smaller $|\mathbf{n}_1^T \mathbf{e}|$, in order to ensure numerical stability. With this, a valid

input : Quadric cost \mathbf{Q} and neighboring planes \mathbf{P} of an edge
output: Edge contraction point \mathbf{v} and its cost
cost $\leftarrow \infty$;
foreach [0..3]-subset $\mathbf{P}' = [\mathbf{N}^T \ - \mathbf{d}^T]$ of neighboring planes \mathbf{P} **do**
 calculate \mathbf{A} and \mathbf{b} from \mathbf{P}' (Equations (4.15) to (4.18));
 solve \mathbf{y} (Equation (4.14));
 $\mathbf{x} \leftarrow \mathbf{A}\mathbf{y} + \mathbf{b}$;
 if $\mathbf{P}^T \mathbf{x} \geq 0$ and $\mathbf{x}^T \mathbf{Q} \mathbf{x} < \text{cost}$ **then**
 $\mathbf{v} \leftarrow \mathbf{x}$;
 cost $\leftarrow \mathbf{x}^T \mathbf{Q} \mathbf{x}$;
 end
end

Algorithm 1: Bounded edge contraction point optimization

substitution is constructed by

$$\mathbf{A} = \begin{bmatrix} \mathbf{e} \times \mathbf{n}_1 & \mathbf{e} \times \mathbf{n}_1 \times \mathbf{n}_1 \\ 0 & 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -d\mathbf{n}_1 \\ 1 \end{bmatrix}. \quad (4.16)$$

Case $m = 2$: When the subspace is a line, it can be constructed by

$$\mathbf{A} = \begin{bmatrix} \mathbf{n}_1 \times \mathbf{n}_2 \\ 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} -(\mathbf{N}\mathbf{N}^T)^{-1} \mathbf{N}\mathbf{d} \\ 1 \end{bmatrix}. \quad (4.17)$$

Case $m = 3$: For three constraints, the subspace collapses to a single point \mathbf{b} . The minimizer can directly be calculated by

$$\mathbf{x} = \mathbf{b} = \begin{bmatrix} -\mathbf{N}^{-T} \mathbf{d} \\ 1 \end{bmatrix}. \quad (4.18)$$

With these four cases, we can completely solve the general problem from Eq. 4.10. Considering computational efficiency, it is worth to note that all calculations of \mathbf{A} , \mathbf{b} and solving Eq. 4.14 for \mathbf{y} involves only inverting matrices up to dimension three. In total, all of the above cases can be computed using only a few hundred floating-point operations.

4.3.5 Bounding Mesh Algorithm

After deriving a cost function that can guide edge decimation and optimize edge contraction points, we finally give a complete overview of the bounding mesh algorithm. In general, the bounding mesh algorithm takes a greedy approach, similar to

```

input : Mesh  $M$  and tolerance  $\varepsilon$ 
output: Bounding mesh of  $M$ 
foreach edge  $e$  of mesh  $M$  do
     $Q(e) \leftarrow \sum_{\mathbf{p} \in P(e)} \mathbf{p} \mathbf{p}^T$ ;
     $\text{cost} \leftarrow \min_v E(e, v)$  (Algorithm 1);
    Add  $(e, \text{cost})$  to priority queue;
end
while true do
    Pop cheapest  $e$  from priority queue;
     $\mathbf{v}^* \leftarrow \text{argmin}_v E(e, v)$  (Algorithm 1);
    if  $E(e, \mathbf{v}^*) > \varepsilon^2$  then
        | return  $M$ ;
    end
    Contract edge  $e$  to vertex  $\mathbf{v}^*$ ;
    Re-calculate  $Q(e)$ ,  $\text{cost}(e)$  for modified edges (Section 4.3.3);
end

```

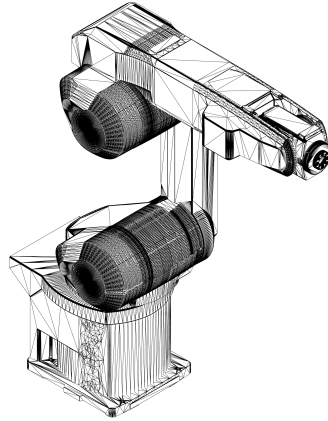
Algorithm 2: Bounding mesh generation

a wide range of iterative mesh simplification schemes [76]. The iterative bounding mesh routine is outlined in Algorithm 2. Apart from the constraints and cost function, this routine is identical to the one in [80]. Estimated costs of edge contractions are first calculated for all edges and stored in a priority queue. After this initialization, the cheapest edges are iteratively contracted until a maximum cost or a target number of vertices is reached.

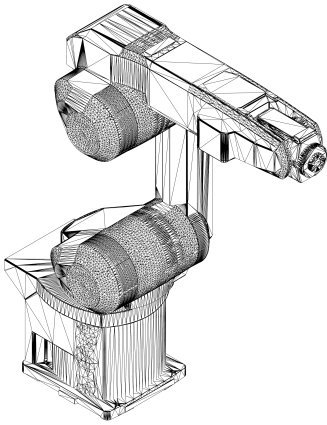
Internally, the bounding mesh algorithm makes several function calls to the bounded edge contraction optimization, which is outlined in Algorithm 1. This procedure essentially follows the closed-form optimization devised earlier in Section 4.3.4, which ensures numerically stable calculation of the cost function minimum and its minimizer, the optimal edge contraction point. For this, Algorithm 1 enumerates the subsets of inequalities and determines the global minimizer. In each subspace, it calculates the minimizer \mathbf{x} by direct geometric solution from Equations (4.15) to (4.18). Finally, Algorithm 1 returns the optimal edge contraction point \mathbf{v} and its cost. The cheapest edge is then contracted until a global distance limit is reached (Figure 4.3). Each contraction triggers the costs of the modified edges being re-calculated, as described in Section 4.3.3.

Evaluation

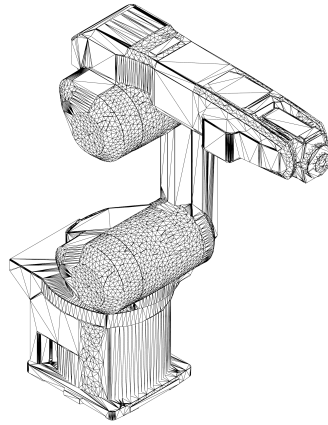
While the bounding mesh algorithm is certainly not restricted to robot task and motion planning, it was developed with the intention to simplify the geometry of



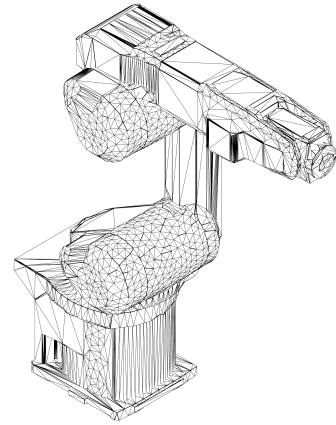
(a) Original mesh with 108,419 vertices



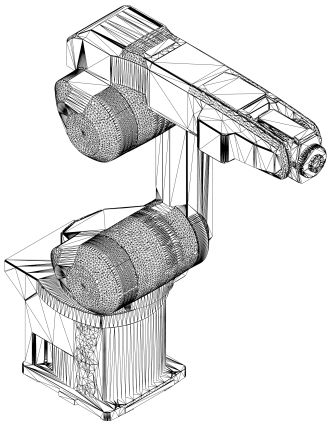
(b) Bounding mesh with 20,000 vertices, $\varepsilon < 0.0034$ m



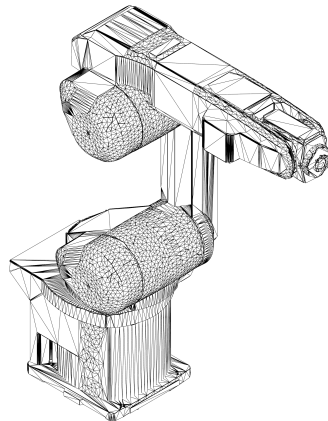
(c) Bounding mesh with 10,000 vertices, $\varepsilon < 0.0062$ m



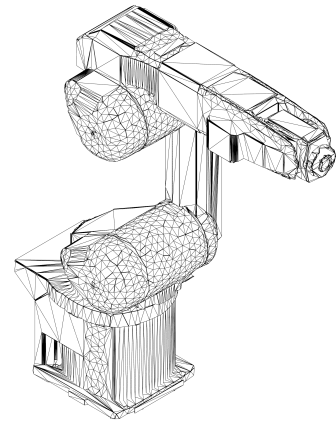
(d) Bounding mesh with 5,000 vertices, $\varepsilon < 0.0090$ m



(e) Inner bounding mesh with 20,000 vertices, $\varepsilon < 0.0028$ m



(f) Inner bounding mesh with 10,000 vertices, $\varepsilon < 0.0046$ m



(g) Inner bounding mesh with 5,000 vertices, $\varepsilon < 0.0088$ m

Figure 4.6: A series of outer and inner bounding meshes of a MITSUBISHI robot.

Table 4.2: Bounding meshes in both directions of several geometric models of industrial manipulators. A quadric cost of plane distances with adjustment of the constant term is optimized (Section 4.3.3).

Number of vertices n	Distance from original to approximation [mm]			Distance from approxima- tion to original [mm]		
	min	mean	max	min	mean	max
<hr/> MITSUBISHI, original triangulation, bounding box diagonal: 1239.37 mm 108 419 (Figure 4.6)						
Outer bounding meshes						
20 000	0	0.082	1.795	0	0.262	3.413
10 000	0	0.222	3.018	0	0.580	6.215
5 000	0	0.546	5.858	0	1.239	9.048
2 500	0	1.284	12.522	0	2.595	20.084
Inner bounding meshes						
20 000	0	0.060	1.788	0	0.333	2.854
10 000	0	0.167	3.039	0	0.739	4.634
5 000	0	0.419	5.071	0	1.642	8.809
2 500	0	0.943	10.788	0	3.809	17.377
<hr/> KUKA, original triangulation, bounding box diagonal: 3443.75 mm 251 117 (Figure B.1 on page 136)						
Outer bounding meshes						
20 000	0	0.636	7.685	0	0.934	22.871
10 000	0	2.066	18.070	0	2.733	24.920
5 000	0	4.900	30.004	0	5.890	52.727
2 500	0	10.793	75.640	0	12.527	81.235
Inner bounding meshes						
20 000	0	0.428	7.537	0	1.891	26.012
10 000	0	1.462	12.643	0	6.541	141.080
5 000	0	3.442	23.578	0	12.985	153.547
2 500	0	5.565	52.924	0	33.452	229.243
<hr/> COMAU, original triangulation, bounding box diagonal: 3719.08 mm 241 190 (Figure B.2 on page 137)						
Outer bounding meshes						
20 000	0	0.335	6.704	0	0.984	18.010
10 000	0	1.320	15.021	0	3.223	43.363
5 000	0	3.672	23.489	0	6.547	49.266
2 500	0	7.909	42.383	0	11.053	70.658
Inner bounding meshes						
20 000	0	0.330	7.681	0	1.328	14.319
10 000	0	1.439	14.056	0	4.284	37.302
5 000	0	3.309	28.341	0	8.521	52.652
2 500	0	5.509	60.743	0	14.868	95.682

robots and objects conforming to the bounded geometric predicates, as defined at the beginning of Section 4.1. We therefore focus our evaluation on the type of model that is usually the most complex in this domain, the robot manipulator itself. First, we discuss the quality of bounding meshes of several robot geometries in terms of the Hausdorff distance metric. After that, we briefly compare the different types of cost functions on a larger set of shapes.

Figure 4.6 shows a typical industrial manipulator with several outer and inner bounding meshes. The original triangulation has 108,419 vertices. As typical, triangles are unevenly distributed, with many thin triangles on curved or filleted surface regions. The bounding mesh algorithm can simplify this type of mesh to a fraction of the number of vertices while assuring a distance of a few millimeters. Even for only 5,000 vertices to represent the whole robot model, the bounding meshes remain closer than 0.01 m to the original, whose height is roughly 1 m. It is worth noting that the robot geometry was not pre-processed for this experiment, it contains a number of small openings and threaded holes, and many triangles with high aspect ratio, which is typical for meshes exported from computer-aided design software. Distances were measured as Hausdorff distances between a high-quality triangulation and the respective bounding mesh. The current version of the implementation performs at the order of 10,000 edge decimations per second on a 2.8 GHz desktop computer. Its memory requirement scales proportionally to the complexity of the input mesh, with approx. 25 MB of memory for each 10,000 vertices.

More detailed results for different types of robot geometries are shown in Table 4.2. In this evaluation, triangulated meshes from three robots from different manufacturers are simplified to both inner and outer bounding meshes. Again, meshes were not pre-processed, so all three contain small openings, thin triangles, and other typical imperfections. The bounding mesh algorithm was configured to minimize plane distance costs and approximate costs by quadrics with adjustment of the constant term (Section 4.3.3). Distances were measured in both directions from and to the original mesh to show both components of the Hausdorff distance individually. Bounding meshes of the first example, the MITSUBISHI robot, are depicted in Figure 4.6; the results of the KUKA and COMAU robot meshes can be found in Appendix B.1. As discussed earlier, the MITSUBISHI robot can be compressed to millimeter accuracy at a fraction of its vertex count. The second model, the KUKA robot, is an example for a very detailed model, as it includes many fine, round cable structures. Compared to the simpler MITSUBISHI mesh, its bounding meshes require more vertices to guarantee distance limits. To ensure a Hausdorff distance of less than 1% of the model size, 10,000 vertices are required for an outer bounding

mesh. For the KUKA model with its many thin structures, inner bounding meshes require roughly double the numbers of vertices to provide a similar Hausdorff distance as outer bounding meshes. The third model of the evaluation, the COMAU robot, possesses thin brackets at its tool shaft, but is otherwise of average geometric complexity, comparable to the MITSUBISHI robot. With regard to its larger size, the approximation results are good, with roughly 11,000 vertices needed to maintain 1% precision for an outer bounding mesh.

For all three robot geometries, the bounding mesh algorithm provides a single-sided approximation with 1% precision at one tenth of the vertex count. Because the discussed geometry models originate from different manufacturers, and other trials with additional meshes did not reveal any further issues, we are sure that the bounding mesh algorithm can generally approximate a robot geometry by a fraction of its vertex count at high precision.

Based on our evaluation, we formulate the following result.

Proposition 1. *The bounding mesh algorithm can reduce most robot geometries to an enclosing single-sided approximation at the order of 4,000–12,000 vertices within a precision of 1% of the diagonal of the geometry. In relative numbers, most triangulations from CAD software can be compressed by a factor of 10–20 at this precision. The resulting complexity depends on the shape; smoother shapes allow higher compression.*

To complete our evaluation of the bounding mesh algorithm, we also compare different cost functions and methods for adding costs after a decimation. For this, we measure Hausdorff distances of the bounding meshes of a larger set of shapes under different cost functions. As cost functions, we test the simple sum-of-point-distances $E_{\text{triangles}}$ and the previously described E_{planes} functions, both with and without adaptation of the constant term of the quadric. All measurement results are listed in Appendix B.2. For all shapes, the cost function E_{planes} provides slightly better approximations than $E_{\text{triangles}}$. The adaptation of the constant does not lead to significantly different results for both cost functions. This result applies to both outer and inner bounding meshes. Besides differences between cost functions, the vertex–distance graphs (Appendix B.2) also show that inner bounding meshes generally require more vertices than outer bounding meshes at the same precision. Furthermore, we can observe that 3D scans can be decimated particularly well; both the STANFORD BUNNY and the DRAGON scans can be compressed even higher than the triangulated meshes.

Limitations

Even though the bounding mesh algorithm was thoroughly tested on a number of real robot geometry models from different manufacturers, it is still worth discussing its limitations. Some limitations are due to the general design of the algorithm; other issues may only arise in case of degenerate input meshes and are amenable to future improvements of the implementation.

Degenerate meshes are generally an issue for geometric algorithms, with some taking precautions or performing additional checks to handle special cases. In general, the bounding mesh algorithm assumes that the input mesh is a triangulated, orientable 2-manifold, or a disconnected set of multiple such meshes. If the input contains a set of meshes, the algorithm takes no advantage of merging overlaps, but rather treats all connected meshes independently. It will distribute costs over connected components to minimize the total costs, which may be a desirable behavior for a robot geometry with multiple rigid bodies. The most important property of this assumption is that the given mesh must be closed. Our implementation does accept open meshes, but it will avoid modifying the border. Strictly speaking, collision and inclusion predicates are no longer defined for open meshes. However, there are many practical examples where meshes of industrial robots contain tiny holes, and this strategy succeeds in constructing a bounding mesh that allows a correct, ε -precise bounding convex decomposition, and bounded geometric predicates will work as if these holes were closed.

Even when input meshes fulfill all specifications, some qualitative properties may degrade performance. Since the proposed bounding mesh algorithm performs edge contractions in a greedy strategy, it does not improve the topology of a given mesh, as opposed to global or re-meshing strategies. Vertices of high degrees (vertices with high numbers of adjacent edges) affect the efficiency of the algorithm because of the high number of inequalities in the edge contraction optimization (Eq. 4.2). An example for a high-degree vertex would be the apex of a triangulated cone. Also, triangles with extreme aspect ratios may affect the quality of the simplification. When multiple such triangles appear in a neighborhood, especially when their orientations differ, edge contractions become forbiddingly expensive. In practice, triangulation in state-of-the-art computer aided design systems is aware of these qualitative features and can produce meshes suitable for bounded decimation.

Distance Limiting Cost Functions

In order to generate bounding meshes within a given distance limit ε , the cost function in Algorithm 2 needs to fulfill two requirements: First, it must not be

smaller than the Hausdorff distance. Second, it needs to preserve this property after an edge contraction. Two types of quadric cost functions are candidates for this set of requirements, the sum of squared distances to planes as in Equation 4.4, denoted as E_{planes} , and the sum of squared distances to circumcenters of triangles, denoted as $E_{\text{triangles}}$.

E_{planes} is an upper bound to the point–mesh distance *if no acute dihedral angles* are present. One can easily verify that if all dihedral angles are obtuse, the sum of squared distances to neighboring planes is greater than the squared distance to the mesh. An alternative to achieve this property is to add plane distance costs to all sharp edges, perpendicular to the edge normal [72, p. 83]. After these quadric costs have been added in the initialization of Algorithm 2, addition of quadrics in modification steps will preserve the upper bound. For the cost function $E_{\text{triangles}}$, it is obviously an upper bound to the point–mesh distance for all meshes, albeit a weaker one. In addition to the distance limitation, it has the notable property that the *group of convex meshes is closed under the bounding mesh operation with cost function $E_{\text{triangles}}$* . (For a detailed discussion, see Appendix A.2.) In other words, for a given convex mesh, a bounding mesh decimation with the cost function $E_{\text{triangles}}$ will result in a convex bounding mesh. Apart from the invariance to position, orientation, scale, and symmetry, this convexity invariance is an interesting property of this variant of the algorithm.

Conclusion

In conclusion, the bounding mesh algorithm can simplify arbitrary meshes with a purely single-sided approximation. While the main motivation for our research in mesh simplification is to speed up robot collision checking and related queries without risking false negatives, the algorithm is not limited to robotics and more generally applies to computer geometry. For bounding mesh edge contractions, we propose a quadric cost function, and an adapted cost function with slight advantages towards convex meshes. Our evaluation indicates that triangulated robot geometries can be simplified to a fraction of their vertex count at low distance errors. In all cases, the approximation is guaranteed to be single-sided, generating outer or inner bounding meshes.

4.4 Bounding Sets of Convex Polyhedra

While a bounding mesh is already a useful mesh simplification by itself, it becomes a particularly powerful approach to efficient geometric queries when combined with

convex decomposition. The bounding mesh simplification effectively reduces the vertex count of complex meshes and guarantees to enclose the original mesh, which is required for safe geometric queries. However, its output is generally non-convex for non-convex input, and therefore inefficient to process in collision and inclusion checking. On the flipside, new convex decomposition techniques were recently developed that can segment non-convex input models into a small set of convex shapes [91, 92]. Convex decomposition alone does not reduce complexity in terms of vertex count, it only segments vertices into approximately convex clusters, and only a small number of vertices are removed in subsequent convex hull operations. Obviously, the combination of convex decomposition and the bounding mesh algorithm can both decompose and simplify arbitrary meshes and generate an efficient structure for collision and inclusion queries, a *bounding set of convex polyhedra*.

Definition 3 (Bounding set of convex polyhedra). *For a given mesh M , we may create bounding sets of convex polyhedra M' with the following properties.*

1. $M' = \bigcup_n \text{convex}(V_n)$, it is a union of convex hulls.
2. $M' \supseteq M$, it encloses the original model.
3. M' is a simplification of M ; it should have fewer vertices and fewer convex polyhedra than an exact decomposition.

Furthermore, we may characterize M' as ε -precise if the Hausdorff distance $d(M, M')$ is no larger than a given parameter ε .

In this section, we will first discuss the state of the art in convex decomposition. Then, we present a method for generating bounding sets of convexes, evaluate our approach in several examples, and discuss its general properties.

4.4.1 Algorithms for Convex Decomposition

Decomposing a non-convex mesh into a small set of convex polyhedra is a challenging problem. Chazelle et al. have shown that an exact decomposition into a minimal set of convex meshes is NP-complete [93] by constructing a mesh whose decomposition solves a given Boolean satisfiability problem. Despite this inherent difficulty, several heuristics have been proposed for decompositions into a non-minimal number of convexes [93, 91, 92]. In a more recent work, Lien and Amato argue that exact decomposition necessarily generates high numbers of clusters, and recommend to relax the problem to an approximately convex segmentation [91, 94]. Developing a hierarchical, approximate algorithm, Lien and Amato demonstrate that even a

slight relaxation can dramatically reduce the number of segments generated. Their algorithm identifies concave regions and partitions these through planar cuts, and follows a divide-and-conquer approach until all partitions' residual concavities satisfy an approximation parameter. Soon after Lien and Amato's pioneering work, Mamou and Ghorbel recognize that planar cuts offer only limited choices in the bisection step, and devise a more general convex clustering based on decimation of the dual graph [92], named HACD. Essentially, Mamou and Ghorbel's approximate convex decomposition decimates edges on the dual graph of triangles in order to identify nearly convex clusters. The decimation follows a weighted cost function of a concavity and an aspect ratio measure. The aspect ratio cost is designed to dominate the first few iterations and to quickly simplify the dual graph; after that, the concavity measure leads the segmentation by design. Evaluation shows a clear improvement over Lien and Amato's approach [92]. While the implementation of HACD applies a mesh simplification strategy that does not preserve the volume of the input mesh, we can easily replace this processing step by our bounding mesh simplification for our application of *bounding* convex decomposition. An alternative, sampling-based approach is proposed by Asafi et al. [95]. Their method is to cluster segments of points that are mutually visible, and is geared towards more general shapes and point clouds.

Very recently, Mamou developed a variant of the hierarchical approximate convex decomposition whose segmentation operates on a voxel grid, named V-HACD [96]. Compared to the original HACD approach [92], it seems to operate faster on detailed meshes, and more robustly on meshes with high genus. Both HACD and V-HACD have been shown to generate results superior to existing methods in terms of accuracy and numbers of convex segments [92, 96], but also stand out in computational speed, and can therefore be regarded as the state of the art in approximate convex decomposition.

4.4.2 Bounding Convex Decomposition

With the above algorithms for approximate convex decomposition and our bounding mesh algorithm, it is straightforward to generate bounding sets of convex polyhedra with the properties in Definition 3. In order to ensure the boundedness criterion, $M' \supseteq M$, we need to modify all mesh simplification steps and apply the bounding mesh algorithm instead, and configure convex decomposition to generate full convex hulls rather than lower-vertex heuristics. We can then generalize the bounding set of convex polyhedra approximation to the following procedure, which applies to a group of convex decomposition algorithms [92, 96, 91]:

1. For convex decomposition algorithms that are susceptible to high vertex counts [92, 91], we pre-process the input to a closely approximating bounding mesh, at distance at the order of $\varepsilon/10$. (V-HACD [96] scales well with high numbers of vertices, allowing this step to be omitted.)
2. We perform the approximate convex decomposition as in [92, 96, 91], with parameters set to disable all further mesh decimation and to output full convex hulls or convex segments.
3. Finally, we simplify the convex decomposition with the bounding mesh algorithm up to a desired approximation distance.

4.4.3 Evaluation

In order to measure the quality of this approach, we first consider the results of the bounding set of convex approximation of the MITSUBISHI model, the same model as in the bounding mesh discussions. After this practical example, we try to identify more general properties of bounding sets of convex polyhedra and discuss quantitative bounds on ideal bounding decompositions.

Figure 4.7 shows the experimental results for the MITSUBISHI example. In robot motion planning, the usual approach is to approximate rigid bodies by convex hulls (Figure 4.7b). While this method is simple, reliable, and allows efficient queries, it cannot guarantee any distance limits. Even though the example model is rather well suited for convex hull approximation with its smooth shape and few concavities, this approximation introduces a large distance error of 53.6 mm (4.3% of the height of the model). Besides this, the convex hull operation can only simplify vertices in concave regions, leaving a large fraction of the original complexity, with a reduction by a factor of 5 in the example. Bounding sets of convex polyhedra can be seen as a generalization to a union of multiple such convex hulls, which is further simplified in terms of the vertex count. In the example in Figure 4.7c, it is evident that most shapes allow a tight approximation by few convex hulls. Furthermore, the number of vertices is reduced dramatically, for instance by a factor of 115 in Figure 4.7d. All this approximation comes at a reasonable precision of 35 mm (2.8% of the height of the model). Since this distance can be controlled by parameters of the algorithm, the properties in Definition 3 are fulfilled.

The resulting bounding convex decompositions shown in Figure 4.7 are only part of a larger evaluation, whose results are summarized in Table 4.3. In order to investigate the complexity and accuracy of the approximation of the bounding convex decomposition more generally, we measure these values under variation of

all relevant parameter settings. The decomposition is performed by V-HACD [96], whose most influential parameters are resolution of voxel sampling, recursion depth, a residual concavity limit c , and a parameter α to balance between decomposition and convex hull operations. The first two parameters and α can trivially be set for optimal accuracy at the expense of offline computation time, while parameter c strongly affects the final number of convex polyhedra. For the bounding mesh algorithm (Section 4.3.5), there exist only the target number of vertices n as a parameter and a discrete choice of a cost function. Since the goal is to approximate convex polyhedra, we choose the distance-limiting cost function $E_{\text{triangles}}$, as defined in Section 4.3.5, which preserves the convexity of the given mesh.

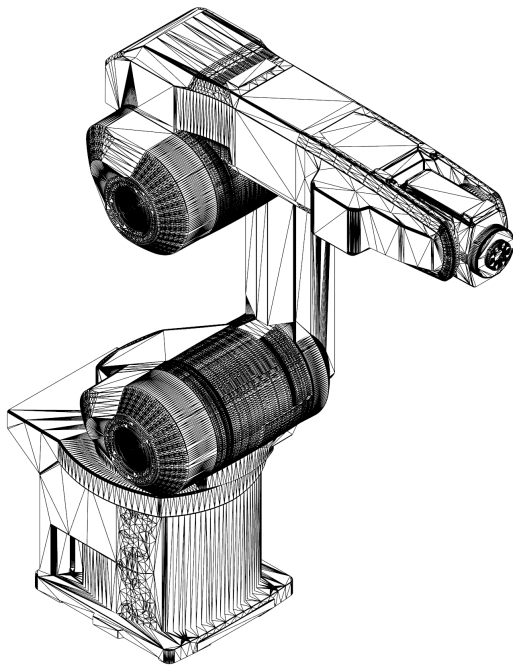
From Table 4.3, we observe that convex decomposition is already an approximation superior to simple convex hulls, both in terms of error distance and complexity. Subsequent bounding meshes reduce the vertex count by more than one order of magnitude, and at a negligible error. For V-HACD, there exists a “sweet spot” at $c \approx 10^{-4}$ to generate a modest number of convex segments at a low distance. For the bounding mesh approximation, an additional distance limit of $\varepsilon \approx 0.005$ m reduces the vertex count by a factor of ten. Since these two steps are independent, future refinements of V-HACD—which currently introduces the larger part of the distance error—will directly enhance the overall quality. A further property of the bounding set of convex polyhedra approach is the low maximum number of vertices per convex body. In applications where worst-case complexity is critical, such as real-time collision avoidance, only low *maximum* numbers can guarantee performance, and these are dramatically lower than in previous approaches. This worst-case benefit will further be shown in a path planning experiment below.

In total, the bounding set of convex polyhedra approach is a powerful geometric simplification, achieving very low vertex counts, lower than conventional convex hulls or convex decomposition alone, and at lower distance errors.

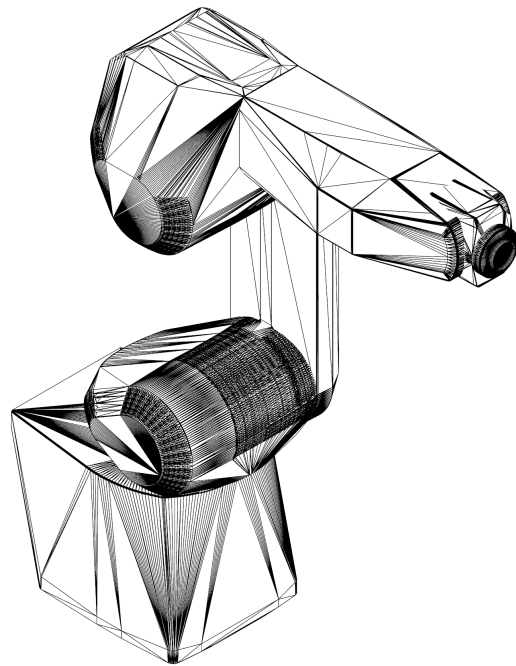
Theoretical Accuracy

While experiments with robot geometries are representative for the types of geometric shapes in our application to efficient robot task planning and show the effectiveness of our approach, it is worth taking a step back and analyzing the general properties of bounding sets of convex polyhedra.

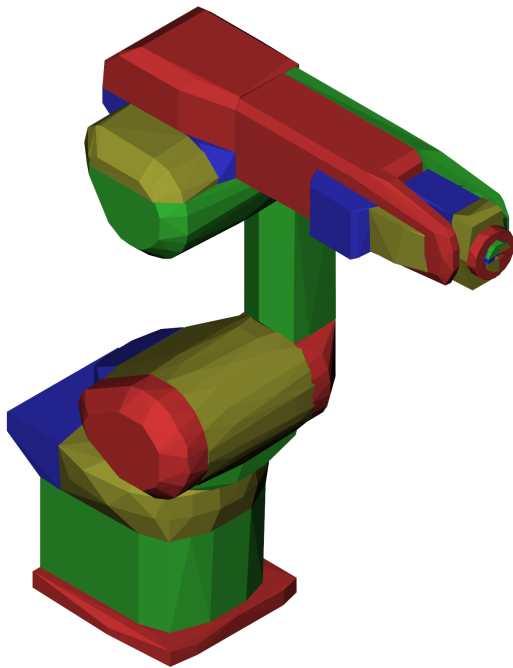
A bounding convex decomposition can be seen as a compromise between the number of convex polyhedra n , their number of vertices $|V|$, and the Hausdorff distance ε to the original mesh. As with probably all mesh simplification, it is hard to quantify theoretical bounds of what simplification is feasible. We cannot



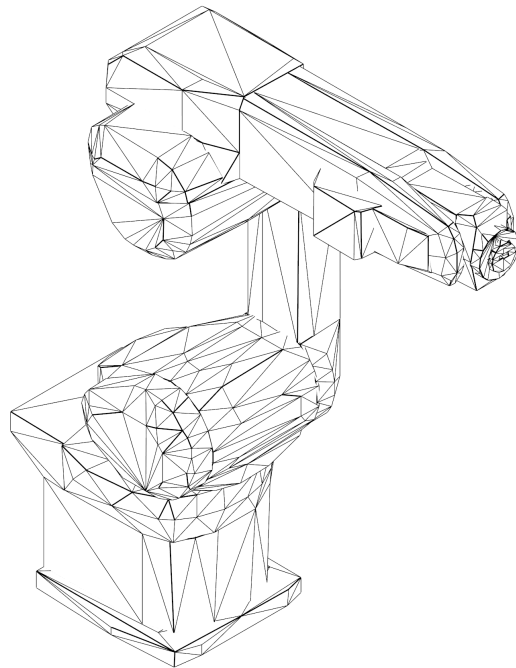
(a) Original triangulation with 108,419 vertices



(b) Convex hulls of all rigid bodies, 7 convex bodies, 20,147 vertices



(c) Bounding set of convex polyhedra, convex segments highlighted in color



(d) Bounding set of convex polyhedra, 64 convex bodies, 942 vertices

Figure 4.7: Bounding set of convex polyhedra of a MITSUBISHI robot. Convex hull approximation of rigid bodies, the usual approach in robot motion planning, only simplifies concave regions (b). A bounding set of convex polyhedra is a more accurate and simpler generalization (c,d).

Table 4.3: Bounding convex decomposition results of the MITSUBISHI model under different decomposition and bounding mesh approximation parameter settings. Parameters c and α refer to V-HACD [96], n is a parameter of Algorithm 2.

Number of vertices n	Number of convexes	Vertices per convex body			Distance from orig- inal to approxima- tion [mm]			Distance from approximation to original [mm]		
		min	mean	max	min	mean	max	min	mean	max
Original triangulation (Figure 4.7a)										
108419	n/a									
Convex hulls of rigid bodies (Figure 4.7b)										
20147	7	1216	2878.14	6119	0	5.35	53.61	0.000	0.001	0.009
Approximately convex segmentation, $c = 10^{-2}$, $\alpha = 10^{-3}$										
9274	30	43	309.13	966	0	2.41	34.93	0.003	1.33	12.38
Bounding sets of convexes thereof										
6574	30	29	219.13	682	0	2.41	34.93	0.003	1.39	12.38
4608	30	23	153.60	469	0	2.42	34.93	0.003	1.43	12.38
2888	30	18	96.26	270	0	2.45	42.30	0.003	1.54	12.38
1700	30	14	56.66	147	0	2.53	42.01	0.003	1.77	12.38
620	30	10	20.66	42	0	3.07	42.30	0.020	3.57	17.79
427	30	10	14.23	28	0	3.92	42.30	0.020	6.46	42.39
Approximately convex segmentation, $c = 10^{-4}$, $\alpha = 10^{-3}$										
9085	64	8	141.95	805	0	2.47	34.97	0.001	1.37	3.50
Bounding sets of convexes thereof (Figure 4.7d highlighted in bold)										
6729	64	8	105.14	594	0	2.47	34.97	0.001	1.41	3.50
3266	64	8	51.03	244	0	2.51	34.97	0.001	1.51	6.87
2030	64	8	31.71	134	0	2.59	34.97	0.001	1.69	7.58
1326	64	8	20.71	73	0	2.75	35.19	0.001	2.09	7.89
942	64	8	14.71	41	0	3.20	35.19	0.031	3.11	19.66
745	64	8	11.64	26	0	4.04	35.19	0.048	4.86	43.04
Approximately convex segmentation, $c = 10^{-6}$, $\alpha = 10^{-3}$										
10993	105	8	104.65	966	0	2.35	34.93	0.004	1.26	2.79
Bounding sets of convexes thereof										
7937	105	8	75.59	682	0	2.35	34.93	0.003	1.29	2.79
5760	105	8	54.85	469	0	2.36	34.94	0.002	1.32	3.96
3824	105	8	36.41	270	0	2.38	34.94	0.001	1.38	5.05
2495	105	8	23.76	147	0	2.47	34.94	0.003	1.57	5.05
1764	105	8	16.80	79	0	2.63	34.94	0.002	1.94	8.14
1171	105	8	11.15	28	0	3.83	34.94	0.016	3.93	42.39

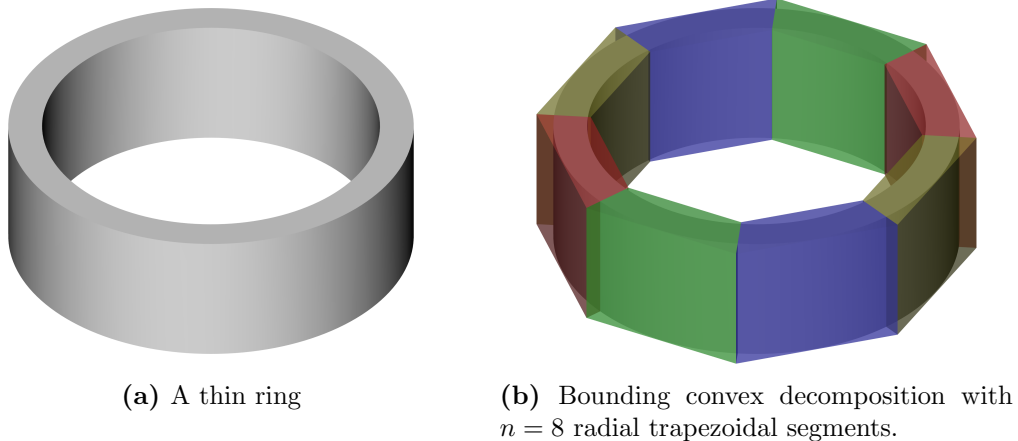


Figure 4.8: For very simple shapes, such as a thin ring, the optimal bounding convex decomposition can be constructed. With increasing number of convex polyhedra n , the approximation converges quadratically.

even guarantee that bounding convex decomposition will simplify any given input mesh, as specified by Definition 3—in the worst case, input meshes such as a set of disconnected tetrahedra or an already processed decomposition can no further be simplified. Apart from empirical evaluation, only the most simple shapes allow derivation of direct limits on the relation of $|V|$, n , and ε . An example for such a shape is a ring with negligible width and unit radius. For symmetry reasons, the best decomposition of a thin ring is a convex approximation of n equal radial segments. For large n , trapezoidal prisms are obviously the best approximation, with a total vertex count of $8n$, and a Hausdorff distance of $1 - \cos(\pi/2n) \approx (\pi/2n)^2$. Figure 4.8 shows an example of a bounding convex decomposition with $n = 8$. The approximation to the concave inner surface of the ring can only be improved by larger numbers of convex bodies.

In a later discussion of swept volumes of robot motion (Section 4.5), we will see that the approximation distance decreases quadratically with the number of convex polyhedra being used. While this gives some indication that smooth shapes can be approximated by bounding convex decomposition, empiric evaluation shows that the same applies to a wider range of shapes.

Bounding Sets of Convex Polyhedra for Collision Detection

In this section, we briefly digress to the more general question whether bounding sets of convex polyhedra can guarantee a better *worst-case computation time for geometric queries* than general meshes or convex hull approximations. Of course,

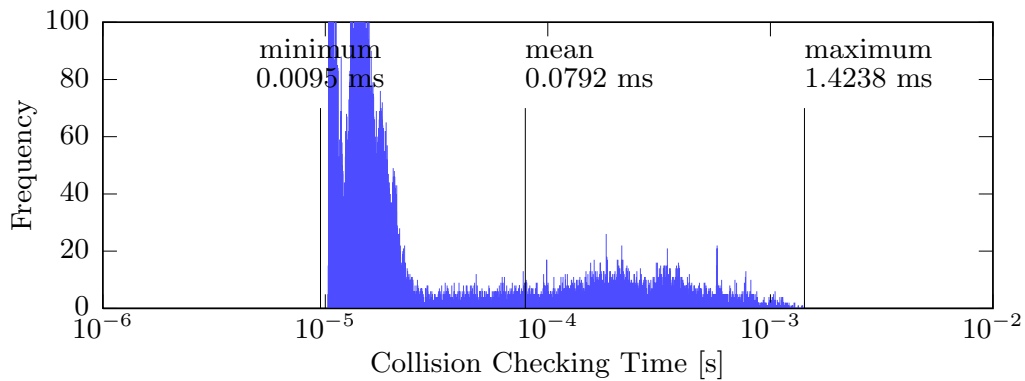
this question is of great interest for a wide range of online motion planning, collision avoidance, and collision checking applications. Many systems have to guarantee collision avoidance within a real-time control cycle. While general mesh collision checking builds on very fast broad-phase algorithms to prune candidate sets and provide good average-case computation times, its worst-case computation time is prohibitively high for applications with real-time requirements.

Proposition 2. *Bounding sets of convex polyhedra allow collision detection (and distance queries) within real-time bounds for practical scenarios.*

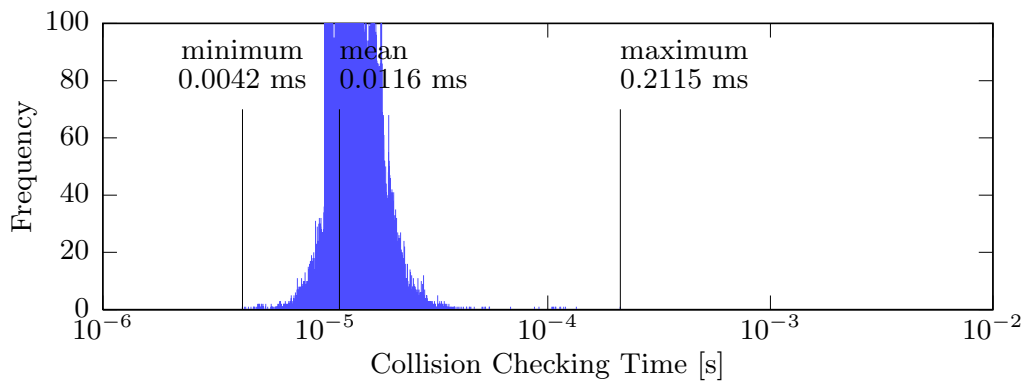
In order to compare the distribution of computation times for collision checking queries for general meshes, convex hull and sets of convex polyhedra approximations, we choose a robot path planning scenario. For path planning and collision checking algorithms, we select the RRT-Connect2 [97] and Open Dynamics Engine [98] implementations, which have been shown very efficient in practical scenarios [24]. Rather than choosing simple algorithms that are possibly easy to speed up, we conduct the experiment in a state of the art path planning system to see the performance impact of the sets of convex polyhedra approximation *orthogonal to all broad-phase heuristics in a modern collision library*. As a scenario, we choose a pick-and-place task with a six-axes industrial manipulator and two narrow passages, the same as analyzed by [99, p. 1]. Computation times were measured on a 2.8 GHz processor running a real-time Linux kernel. Durations of collision checks were collected over 10 planned paths, all of which could be solved for all three geometries. We purposely choose a modern RRT-Connect path planner for our collision query benchmark because optimized planners may potentially focus on narrow passages and make collision queries that are very different from random ones.

To discuss our hypothesis that an individual collision check on a set of convex polyhedra has real-time capable worst-case behavior, we consider the distribution of the duration of single checks, rather than the duration of the full path planning procedure. Measurement results are shown in Figure 4.9, with a clear indication that convex bodies allow faster collision checks, particularly in the worst case. In addition, there is another significant observation that bounding sets of convex polyhedra allow even better worst-case timing. An explanation for this may be their substantially lower vertex count. All in all, results indicate that bounding sets of convex polyhedra provide a *more predictable* performance, which is suitable for real-time requirements in the worst case. In this benchmark, worst-case computation time is improved by an order of magnitude.

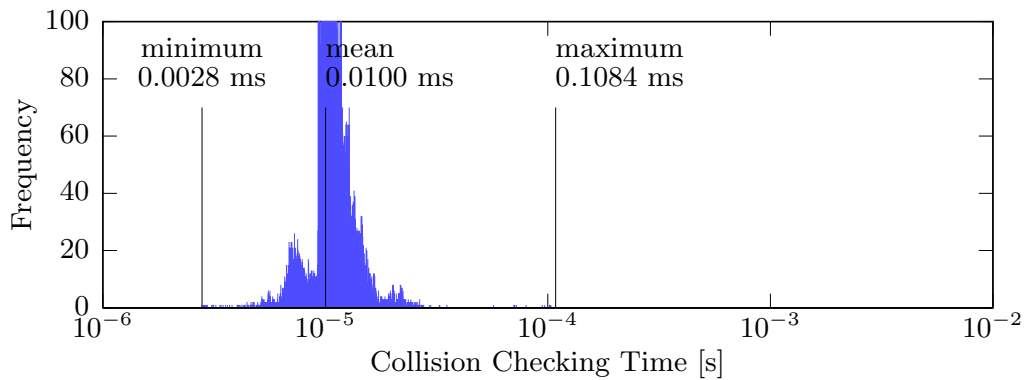
An even stronger argument is the theoretical worst-case complexity of concave-concave collision queries compared to pure convex-convex queries. While concave-



(a) Original, non-convex robot mesh, 9,665 vertices, 26 polyhedra



(b) Convex hull approximation, 2,649 vertices, 26 convex polyhedra



(c) Bounding set of convex polyhedra, 438 vertices, 41 convex polyhedra

Figure 4.9: Distribution of computation times for collision checking queries in a typical path planning scenario with respect to robot geometry approximations, using a state-of-the-art collision checking library. Queries with a normal, non-convex robot mesh are in general expensive, and extremely costly in some cases (a) (note the logarithmic time scale). Bounding sets of convex polyhedra allow faster collision checks in the average case, and provide predictable efficiency even in the worst case (c).

concave algorithms can only rely on exhaustive checking of triangle pairs in the worst case, which scales with $\Omega(nm)$ for two shapes of n and m vertices, convex-convex queries are answered by the Gilbert/Johnson/Keerthi (GJK) algorithm [100] in $\mathcal{O}(n+m)$. Under mild assumptions about the number of convex polyhedra in the scene, collision queries on sets of convex polyhedra are efficient in the worst case. As a corollary, distance queries are likewise provided by the GJK algorithm and can also be solved efficiently for sets of convex polyhedra. Further applications of efficient distance queries include various online motion planning and control schemes, such as repulsive force control for haptic devices or potential field techniques for robot collision avoidance.

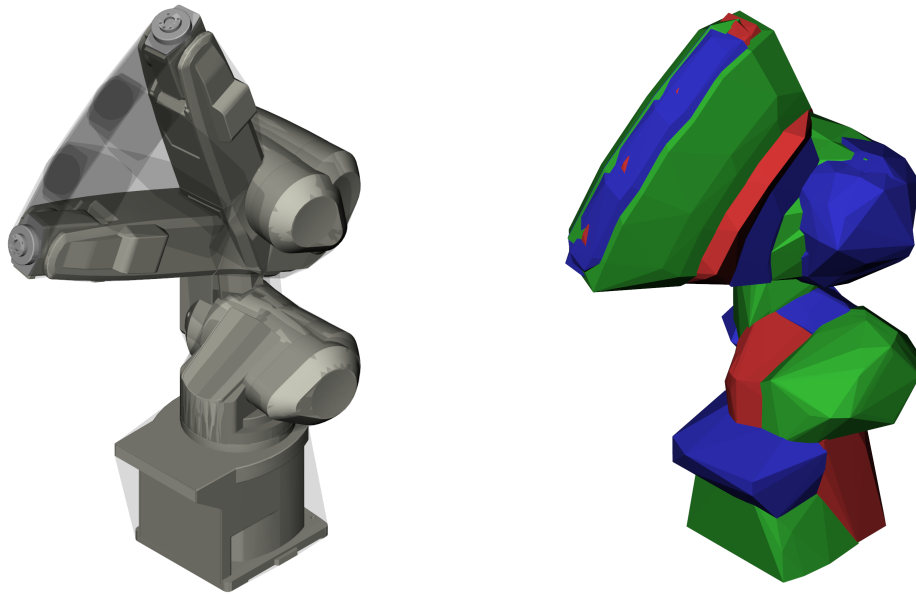
4.5 Bounding Swept Volumes

After our discussion of the general properties of bounding meshes and bounding sets of convex polyhedra, we return to the original goal to evaluate bounded geometric predicates (Definition 1), which are required by the KABouM integrated task and motion planner. In the previous sections, single objects were decomposed into bounding sets of convexes, which allows efficient evaluation of two of the geometric predicates: whether two objects m_0 and m_1 collide, that is $m_0 \cap m_1 \neq \emptyset$, and whether one object is included in another, $m_0 \subseteq m_1$. In the following, we generalize our concept of single-sided approximation to the remaining two bounded geometric predicates, the collision queries between a swept volume of a robot motion and an object, and between two swept volumes of robot motions. Both predicates are defined as ε -precise bounded collision predicates, they must report all collisions exactly, and all non-collisions that are further apart than ε (see truth table on page 44).

As proposed in our previous work [8], the swept volume of a chain of revolute joints can efficiently be approximated by a bounding set of convex polyhedra (Figure 4.10), provided that a bounded convex decomposition of the robot is available.

Proposition 3. *Given is a serial manipulator of k revolute axes and its geometric boundaries as a bounding convex decomposition of m convexes with n vertices each. Then, the swept volume of a length $|Q|$ linear joint space motion can ε -precisely be approximated by a bounding convex decomposition of $\mathcal{O}(km|Q|/\sqrt{\varepsilon})$ convexes of $2n$ vertices each.*

Proof by construction. It suffices to construct the swept volume of an individual convex body of n vertices, as the final result can be obtained from the union of the



(a) Swept volume of serial revolute joints, (b) Bounding set of convex polyhedra approximating a swept volume.

Figure 4.10: The swept volume of a robot motion can be approximated by a small bounding set of convex polyhedra.

resulting m swept volumes. Let Q denote the joint space path, and r_{max} the maximum distance of a vertex to the first axis of the robot, summing up all link lengths and the distance to the furthest vertex. The strategy is to find an adequately large sampling distance Δq along the path that fulfills the ε -distance constraint, which allows us to show quadratic convergence with respect to the number of samples.

For a single revolute joint with an endpoint at distance r , a rotation by an angle $q < \pi$ will let the endpoint deviate from the chord (straight line from start to end) by $\varepsilon = r(1 - \cos(q/2))$ [101]. For general serial kinematics of multiple revolute joints, Baginski [102] describes an upper bound for the deviation of the path of a point from the chord of that path:

$$\varepsilon \leq r_{\max} \left(1 - \cos \left(\sum_i |q_i| / 2 \right) \right) \quad (4.19)$$

Note that this bound, contrary to other solutions like [103], does not assume any limit of the screw motion in each path segment, but only depends on more general properties of serial revolute kinematics. In addition to the trivial case $k = 1$, Baginski's bound is tight when all link lengths but the last one tend towards zero and all joints rotate in the same direction around the same axis.

While Eq. 4.19 can readily be used to solve for Δq , we would like to show its quadratic convergence with respect to the number of samples and obtain Proposition 3. For this, we apply the second order series expansion of the cosine, $\cos(q) \geq 1 - (q^2/2)$, and obtain the quadratic bound

$$\varepsilon \leq r_{\max} \left(\sum_i |\Delta q_i| \right)^2 / 8. \quad (4.20)$$

As a result, one may choose the angular discretization step

$$\Delta q := \sqrt{8\varepsilon/r_{\max}} \quad (4.21)$$

to generate swept volumes at a desired precision ε , and at a quadratic convergence rate.

Similar to Xavier et al. [101], we form convex hulls of consecutive poses of the convex bodies, and add an implicit offset ε as Schulman et al. [103]. Let $q(i)$ be a discretization of the path Q at steps Δq , and let Body be a bounding convex decomposition of the robot geometry comprising m convex sets of vertices. For each sweeping convex polyhedron, we compute its convex hull at subsequent path segments $q(i)$ and $q(i+1)$, applying the forward kinematics function FK .

$$\text{SV}(\text{Body}, Q) = \bigcup_m \bigcup_i \text{conv}(\text{FK}(q(i+1)) \text{Body}_m \cup \text{FK}(q(i)) \text{Body}_m) \quad (4.22)$$

It directly follows that $\mathcal{O}(k|Q|/\sqrt{\varepsilon})$ convex polyhedra, of at most $2n$ vertices each, are sufficient to approximate the swept volume of one convex body at precision ε , where $|Q|$ denotes the length of Q in joint space. Finally, a bounding set of convexes is obtained by an offset surface at distance ε ; rather than computing this offset explicitly, the offset is implicitly taken into account for all future distance queries, as proposed by [103]. \square

Intuitively, Proposition 3 implies that doubling the number of such linear path segments will quadruple the precision of the approximation, but only double the complexity of the swept volume. Figure 4.11 illustrates how a bounding swept volume is generated for a single path segment. Assuming that the path segment is not longer than Δq in joint space, the vertices of a convex body cannot deviate further than ε from the chord of their path (shown as a dotted line). For a single vertex, this region may be regarded as a capsule in 3D space (shown in gray). The swept volume is constructed with an ε -offset of the convex hull of two subsequently sampled positions (shown in blue). In the worst case, all $2n$ sampled vertices will form

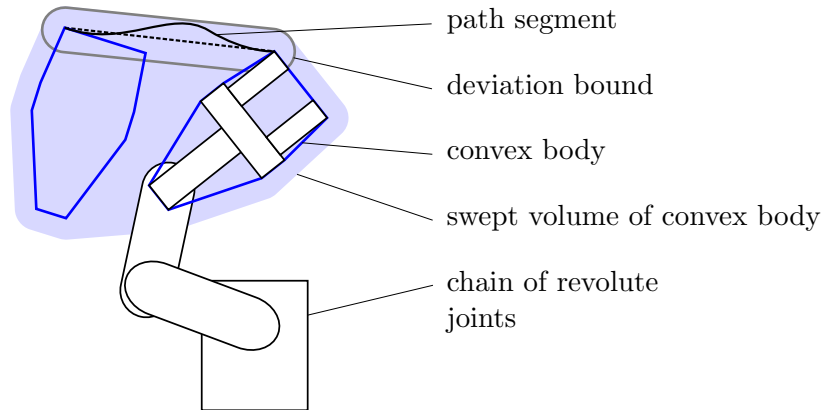


Figure 4.11: The swept volume of a path can be sampled by unions of convex hulls, and the deviation converges quadratically with the number of samples.

the convex hull. Rather than explicitly computing offset surfaces, whose rounded corners cannot be represented by small numbers of vertices, only convex hulls are stored, and the offset distance is implicitly subtracted in all future distance queries. Note that Eq. 4.19 can be made slightly stronger by calculating r_{max} and the subset sums of q for each link individually. In an implementation, this results in fewer samples and simpler swept volumes, especially for the first few links.

In theory, this approach to compute swept volumes is not limited to outer boundary representations. Negating the offset direction, one can generate inner swept volumes, which would help evaluate whether a robot motion is completely included within a certain geometric region. However, this would require an inner convex decomposition of all shapes in the first place, and it is not clear in which task and motion planning domain this type of predicate would be relevant. To summarize, we can generate *bounding swept volumes* of revolute kinematics with rather small sets of convex polyhedra, provided that the robot geometry is already available as a bounding set of convex polyhedra (Figure 4.10b). This geometric representation then allows very efficient queries to the bounded geometric predicates from Definition 1. For static scenes, swept volume generation and continuous collision detection is not necessary. In robot task planning, however, movable obstacles, multiple robots and more general behavior go well beyond the concept of static scenes, and swept volumes are one way to formulate very generic predicates to solve these types of scenarios.

Conclusion

In this chapter, we have developed a set of bounded geometric predicates (Definition 1). In contrast to regular collision and inclusion queries, these predicates allow single-sided tolerances. After discussing the state of the art in mesh approximation, we devised the new bounding mesh algorithm to implement these predicates. In our experiments, we can compress several robot geometries by a factor of 10–20, down to 4,000–12,000 vertices, at a precision of 1%. In a second step, we combined the bounding mesh algorithm with an algorithm for convex decomposition to represent the geometry as bounding sets of convex polyhedra. This approximation permits an efficient evaluation of all geometric predicates and also applies to more general collision detection and distance computation problems. To complete the set of bounded geometric predicates, we presented a method to generate swept volumes for serial revolute kinematics. In context of the KABouM planning system, geometric predicates allow the symbolic planner to evaluate the geometric state. In the next chapter, we will consider the opposite problem and derive an approach to generate geometric states that fulfill symbolic properties, before we can finally evaluate the integrated planner.

Chapter 5

Sampling with Geometric Constraints

As mentioned earlier, there is wide consensus among researchers that robot task and motion planning requires hybrid planning in both symbolic and geometric spaces with systematic mapping between discrete symbols and continuous state vectors [4, 27, 66]. In the previous chapter, a set of geometric predicates was proposed to perform queries from the task planner to the geometric state. A geometric predicate abstracts from the geometric state and returns a discrete value, most commonly a truth value. Besides this geometric–symbolic mapping, it is equally important for a hybrid planner to map from abstract symbols to geometric states by instantiating new geometric states or refining symbols with a geometric meaning. Without such instantiation of geometric states and sampling in the geometric space, the search could never be complete, and would fail at solving the simplest scenarios. A common example to generate a new geometric state would be to sample a robot configuration at random, which is a basic step in many path planning algorithms. Sampling geometric states is an inherently harder problem because symbolic conditions must be fulfilled that, in general, form a *lower-dimensional, non-trivial manifold* of the full geometric space. Therefore, the task and motion planning problem cannot be solved by random sampling of robot configurations and object positions. In almost all scenarios, geometric preconditions of actions form lower-dimensional manifolds, and random sampling has zero probability of generating a geometric state where these actions are applicable. As an example, the subspace of robot configurations where the robot’s gripper holds an object cannot be covered by simple random sampling. In this case, a *constraint sampling* method is required that covers the subspace. In contrast, a collision-free robot configuration can generally be sampled, because the probability for a collision-free pose is greater than zero. This is a simpler case and can be handled by rejection sampling methods.

In general, there exist three sampling strategies relevant to robot task and motion planning [104]. In the simplest case, the constraint manifold has a non-zero volume in the configuration space, and rejection sampling can cover this manifold (Figure 5.1a). This strategy is chosen by most sampling-based path planning algorithms. If the constraint manifold has a lower dimensionality but can be covered by a closed-form algebraic mapping from a Euclidean space, samples can be computed from random points drawn from that Euclidean space (Figure 5.1b). However, only simple manifolds have such simple shape, and even then the mapping function needs to be implemented by hand, for each type of shape. Still, closed-form sampling is often sufficient for grasp planning. Constraint projection sampling is the most general strategy, as it only expects a cost function suitable for iterative projection (Figure 5.1c).

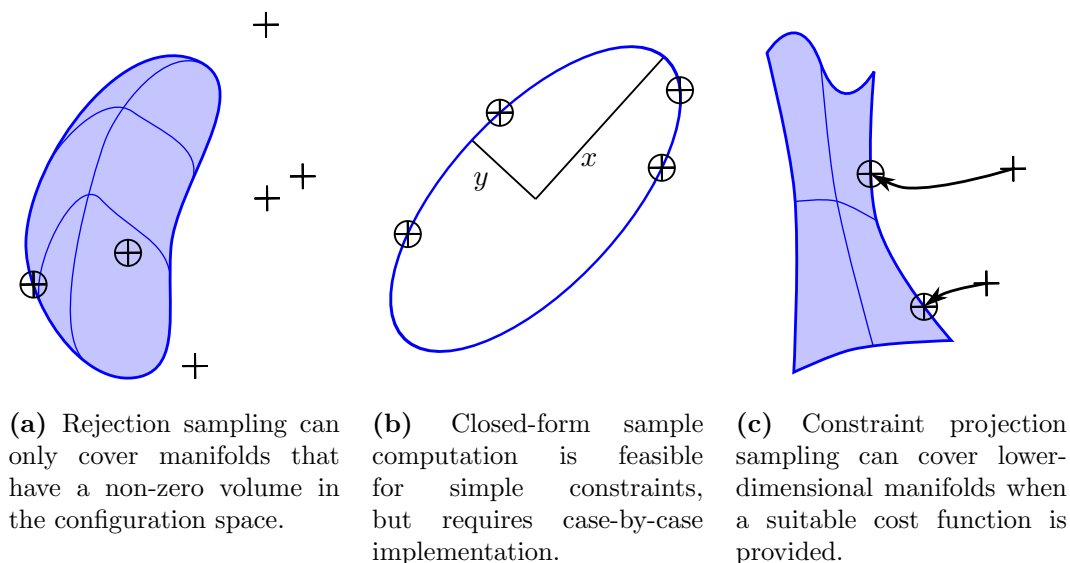


Figure 5.1: Sampling strategies to cover constraint manifolds (adapted from [104, p. 7]).

5.1 Related Work

Task and motion planners follow different approaches to refine symbolic states with geometric sampling. The BHPN planner by Kaelbling and Lozano-Pérez [27] uses the concept of *generator functions* to make choices in the geometric space. Generator functions are very different from our geometric sampling functions, as they apply action-specific heuristics depending on both the initial state and the goal state, while our geometric sampling is more generically defined by constraints. In addition, BHPN performs a regression from the goal state rather than a progression, which naturally requires a different approach to geometric sampling. Apart from direct applications to hybrid planning, constraint sampling is covered by related works in manipulation planning and task-level robot programming.

Manipulation Planning

Sampling in geometric subspaces is a recurring problem in the context of robot manipulation planning. Already in the early days of the Handey robot [105], it was realized that manipulation may generally require a sequence of transits and transfers with re-grasping. More generally speaking, transitions between transit and transfer motions form a lower-dimensional manifold that requires more specific sampling. Similar to our definition of the robot task planning problem, even the manipulation

planning problem cannot be solved by hierarchical calls to a path planner, but requires a larger, combined search space.

A common formulation of the manipulation planning problem was proposed by Alami, Siméon and Laumond in 1989 [106], who suggest a random sampling approach in a combined search space of robot and object configurations [106], [57, p. 332ff.]. This composite search space can then be explored by probabilistic roadmaps, as described by Siméon et al. [107]. In particular, they discuss the problem of finding lower-dimensional transitions between manipulation actions, which are sparse in the search space. For this, they explicitly generate samples in the intersection of the transit and transfer subspaces, which correspond to the space of stable object placements and the space of rigid robot–object connections, respectively.

Barry et al. [108, 54] generalize manipulation planning to more diverse action planning with pushing and sliding actions, and usage of tools, also referred to as multi-modal motion planning. This approach can be seen as a sampling-based solution closely related to task and motion planning. For geometric sampling, it uses projection functions to map random samples to lower-dimensional manifolds. Hauser and Ng-Thow-Hing [34] perform a hybrid search in the mode graph, following modes and lower-dimensional transitions between modes. Here, a mode represents a continuous sub-manifold of the configuration space, such as a moving robot that keeps a single contact state. Hauser and Ng-Thow-Hing demonstrate their Random-MMP planner to generate a trajectory for a bipedal humanoid to walk and push an object on a table. While these two multi-modal motion planners take a task and motion planning approach different from that of our KABouM planner, their proposed techniques for geometric sampling can be applied more generally.

Of course, the field of robot manipulation covers more general questions of navigation among movable obstacles, which can be seen as a pick-and-place task without specified goal locations, for which specific algorithms exist [109, 110], and is also related to grasp planning, dynamics, stability, and friction [111, 57]. However, for our definition of the task and motion planning problem, we limit ourselves to pre-defined manipulation actions with constraint relations between robot tools and objects.

Task-Level Programming

Besides manipulation planning, our approach to sample geometric states in constraint manifolds is closely related to task-level programming [112, 113, 114, 115]. In task-level programming, constraint relations between feature coordinate systems of kinematics and objects can be specified, and are automatically solved for subsequent motion planning. The field of task-level programming dates back to Ambler

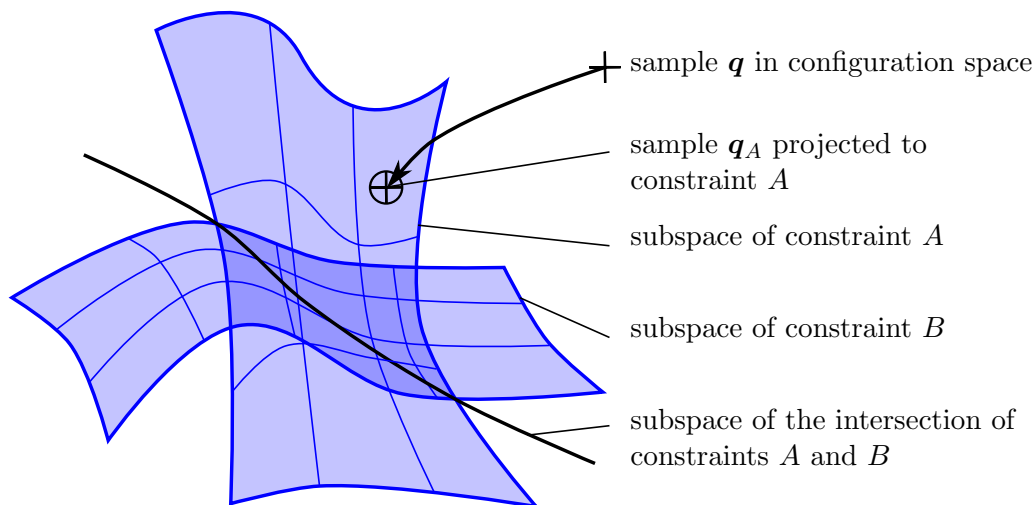


Figure 5.2: The set of conditions of a symbolic action may contain geometric constraints, which, in general, form manifolds lower in dimensionality than the full configuration space. Sampling a geometric state that fulfills one or more constraints therefore requires an iterative solution [54, p. 64f.].

and Popplestone’s spatial relationship solver [116] in 1975. They apply symbolic methods to solve poses between objects that fulfill a system of constraints, and were already motivated by the prospect of more intuitive robot programming. In 1984, Lozano-Pérez and Brooks [114] present the Automatic Task Level Assembly Synthesizer (ATLAS) system, which solves assembly constraints and generates robot trajectories, and achieves a robot programming formulation on the task level, independent from specific kinematics and geometry. Rutgeerts [112] and de Schutter et al. [113] develop the above ideas further and describe the Instantaneous Task Specification using Constraints (iTASC) framework, which generalizes constraint-based robot programming to sensor-based tasks. In the view of geometric sampling, iTASC contributes to the automatic resolution of spatial and kinematic constraints. While iTASC is designed for robot control with online sensing, its constraint formulation in terms of feature coordinates is similar to our geometric constraints, in contrast to formulations that can only directly refer to the operational space of a kinematic.

In our own previous work, we solve geometric constraints to allow task-level programming in terms of CAD semantics [14]. The following problem formulation, cost functions, and algorithm extend and elaborate our earlier work, and apply it to the problem of sampling in constraint spaces.

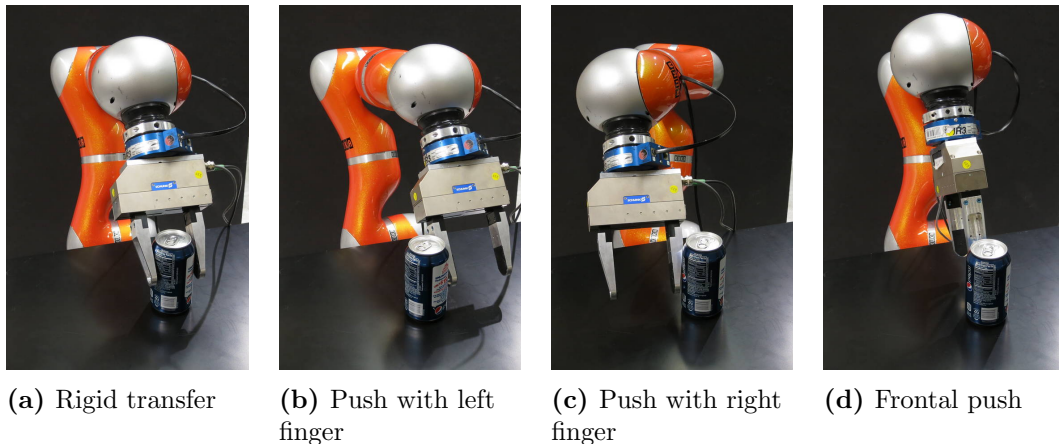


Figure 5.3: The geometric constraint formulation allows different types of robot–object relations to be defined from primitive constraints, similar to mating operations in computer-aided design software. All the above grasp families have a remaining degree-of-freedom around the rotational axis of the object, in addition to a redundant degree-of-freedom of the LBR4 robot. While rigid transfer motions may use all robot joints, push motions follow a one-dimensional Cartesian subspace along the pushing direction [117].

5.2 Geometric Constraint Formulation

A central motivation for our approach to robot task and motion planning is to formulate the problem on the task level, and follow a generic solution to generate intelligent robot actions. For this reason, we define relations between kinematic structures and objects in terms of geometric constraints, which should be fulfilled by sampling certain types of configurations automatically, rather than by providing explicit, manual sampling methods for each type of configuration. While some grasping poses can be defined by a rigid transformation between robot and object, the geometric constraint formulation allows more general types of relations. As an example, Figure 5.3 shows a number of example relations between a parallel gripper and a cylindrical object. When these relations are expressed in terms of geometric constraints, a constraint sampling method can take advantage of the rotational degree of freedom of the object and cover the subspace of grasping configurations, without the need for any domain-specific or manually defined parameters.

In our formulation, a geometric relation is defined as a conjunction of geometric constraints—coincidence, parallelism, distance, or angle—between the primitive shapes points, lines, and planes, which may refer to the tool frame of a kinematic or to a coordinate frame of an object. A list of the types of constraints that are currently covered by our implementation is shown in Table 5.1. This formulation extends our previous work on constraint-based robot programming [14]. While the

presented algorithm and cost functions are mostly an adaptation of our previous work to the problem of sampling in constraint spaces, we add a deeper discussion of completeness and distribution properties, which are of interest to the sampling problem.

Following the notation of the task and motion planning problem definition (Section 3.1), a kinematic structure $\mathbf{R} \in \mathcal{R}$ is characterized by a forward kinematic function FK, which maps from an n -dimensional configuration space to the pose of its tool, $\mathbb{R}^n \mapsto \text{SE}(3)$. To facilitate the formulation of geometric constraints, the tool frame of a kinematic further possesses a set of primitive shapes \mathbf{P} , which contains points, lines, and planes. These features are not necessarily parts of the geometric body, but can more generally refer to tool center points, axes of rotation, or other virtual entities. Similar to kinematic structures, an object likewise possesses a set of primitive shapes that refer to the coordinate frame of that object. This constraint formulation with shape features is more powerful than direct references to the operational space of a kinematic [112, 113]. At the same time, it allows a pure task-level description for some types of geometric relations that would otherwise require manually defined parameters.

5.2.1 Constrained Sampling Problem

Before we attempt to solve geometric constraints in the above formulation, we first define the general properties of a constraint sampling algorithm. Clearly, a constraint sampling algorithm should only produce correct samples that fulfill the given constraints. Using a source of randomness, the algorithm is further required to *cover* the constraint manifold.

Definition 4 (Constraint Sampling Algorithm). *Given a constraint manifold C in a configuration space Q and a cost function $F_C : Q \mapsto \mathbb{R}^c$ whose kernel is C , i.e. $F_C(q) = \mathbf{0} \Leftrightarrow q \in C$, a constraint sampling algorithm is defined as follows:*

1. (Correctness) *It outputs a constraint sample $q \in C$.*
2. (Coverage) *For any given ε -neighborhood within C with $\varepsilon > 0$, subsequent sampling will eventually generate a sample within that neighborhood [118].*

First, we elaborate how such constraints can be defined in terms of geometric relations between robots and objects. This formulation of specific cost functions directly leads to an iterative strategy. At the end of the chapter, we revisit the coverage criterion, which has been proven for a wide range of sample-project strategies by Berenson and Srinivasa [118].

Table 5.1: Definition of geometric constraints and their cost function. \mathbf{p} denotes a point on a shape, \mathbf{d} the direction vector along a line, and \mathbf{n} the normal vector of a plane. This list extends our previous definition published in [14].

Constraint type	First primitive shape	Second primitive shape	Cost function
Coincident	Point \mathbf{p}_1	Point \mathbf{p}_2	$\mathbf{p}_1 - \mathbf{p}_2$
Coincident	Point \mathbf{p}_1	Line $(\mathbf{p}_2, \mathbf{d}_2)$	$\text{null}(\mathbf{d}_2)^\top (\mathbf{p}_1 - \mathbf{p}_2)$
Coincident	Point \mathbf{p}_1	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\mathbf{n}_2^\top (\mathbf{p}_1 - \mathbf{p}_2)$
Coincident	Line $(\mathbf{p}_1, \mathbf{d}_1)$	Line $(\mathbf{p}_2, \mathbf{d}_2)$	$\text{null}(\mathbf{d}_1)^\top [(\mathbf{p}_1 - \mathbf{p}_2) \quad \mathbf{d}_2]$
Coincident	Plane $(\mathbf{p}_1, \mathbf{n}_1)$	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\mathbf{n}_2^\top [(\mathbf{p}_1 - \mathbf{p}_2) \quad \text{null}(\mathbf{n}_1)]$
Parallel	Line $(\mathbf{p}_1, \mathbf{d}_1)$	Line $(\mathbf{p}_2, \mathbf{d}_2)$	$\text{null}(\mathbf{d}_1)^\top \mathbf{d}_2$
Parallel	Plane $(\mathbf{p}_1, \mathbf{n}_1)$	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\text{null}(\mathbf{n}_1)^\top \mathbf{n}_2$
Parallel	Line $(\mathbf{p}_1, \mathbf{d}_1)$	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\mathbf{n}_2^\top \mathbf{d}_1$
Perpendicular	Plane $(\mathbf{p}_1, \mathbf{n}_1)$	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\mathbf{n}_1^\top \mathbf{n}_2$
Distance d	Point \mathbf{p}_1	Point \mathbf{p}_2	$\ \mathbf{p}_1 - \mathbf{p}_2\ - d$
Distance d	Point \mathbf{p}_1	Line $(\mathbf{p}_2, \mathbf{d}_2)$	$\ \text{null}(\mathbf{d}_2)^\top (\mathbf{p}_1 - \mathbf{p}_2)\ - d$
Distance d	Point \mathbf{p}_1	Plane $(\mathbf{p}_2, \mathbf{n}_2)$	$\ \mathbf{n}_2^\top (\mathbf{p}_1 - \mathbf{p}_2)\ - d$

The constrained sampling problem is to obtain a sample $\mathbf{q}_\mathbf{C}$ that fulfills a set of constraints \mathbf{C} , which relates one or more primitive shapes of both kinematic structures and objects [14]. A constraint $C \in \mathbf{C}$ may be described by a cost function $f_C : \text{SE}(3) \times \text{SE}(3) \mapsto \mathbb{R}^c$ that equals the zero vector ($f_C = \mathbf{0}$) if and only if the constraint is fulfilled. With these cost functions, the cost of a kinematic configuration, which may be subject to multiple constraints, simplifies to a single vector-valued function $F : \mathbb{R}^n \mapsto \mathbb{R}^c$ (Eq. 5.1).

$$\underset{\mathbf{q}}{\text{argmin}} \sum_{\mathbf{R} \in \mathcal{R}} \sum_{C \in \mathbf{C}} f_C^2(\text{FK}_{\mathbf{R}}(\mathbf{q}), \mathbf{x}) = \underset{\mathbf{q}}{\text{argmin}} |F(\mathbf{q})|^2 \quad (5.1)$$

Here, $\mathbf{x} \in \text{SE}(3)$ refers to an object pose that is related to the kinematic through a constraint. A cost function f_C depends on the parameters of the primitive shapes \mathbf{P} (Table 5.1), which is a linear function for most types of constraints. However, the parameters of the primitive shapes depend on an object pose \mathbf{x} , or on a robot tool frame $\text{FK}_{\mathbf{R}}(\mathbf{q})$, which ultimately depends on the robot's configuration \mathbf{q} . The set of cost functions for all constraints $C \in \mathbf{C}$ is then written as a function $F(\mathbf{q})$ suitable for iterative sum-of-squares minimization [14].

Similar to earlier approaches to constraint sampling [118, 54], we may sample a random, unconstrained configuration $\mathbf{q} \in \mathbb{R}^n$ and follow an iterative minimization of its cost in order to project it on the constraint space. Since the cost function is


```

input : Set of geometric constraints  $\mathbf{C}$ 
output: Random sample  $\mathbf{q}$  that fulfills constraints  $\mathbf{C}$ 
 $\mathbf{q} \leftarrow \text{RandomSample}() \in \mathbb{R}^n$ ;
cost  $\leftarrow \infty$ ;
while  $\text{norm}(\text{cost}) > \text{epsilon}$  do
    | cost  $\leftarrow F(\mathbf{q})$  (Table 5.1);
    |  $\mathbf{J} \leftarrow \text{CentralDifferences}(F, \mathbf{q})$ ;
    |  $\mathbf{q} \leftarrow \mathbf{q} - \text{PseudoInverse}(\mathbf{J}) \text{ cost}$ ;
end
return  $\mathbf{q}$ ;

```

Algorithm 3: Constraint sampling by iterative projection

rather smooth and its minimum is clearly defined, it can be solved by a simple Gauss-Newton method [14]. As outlined in Algorithm 3, we start with an unconstrained configuration \mathbf{q} that is randomly sampled in the robot's configuration space. In each iteration, cost $F(\mathbf{q})$ and Jacobian matrix $\mathbf{J}_{i,j} = \delta F_i / \delta q_j$ are computed; the current implementation approximates the derivatives by central differences. The Gauss-Newton step performs a linear update on the configuration to minimize costs, using a pseudo-inverse to handle rank-deficient Jacobian matrices. When the cost function converges to zero, we finally obtain a constraint sample configuration \mathbf{q} .

With this constraint sampling algorithm at hand, two issues need to be discussed. First, we need to design an appropriate set of cost functions. Second, it is not obvious whether the above algorithm can cover the constraint space completely.

5.2.2 Design of Geometric Cost Functions

The choice of cost functions has a great impact on the efficiency and stability of the projection algorithm. In order to allow the Jacobian $\mathbf{J}_{i,j} = \delta F_i / \delta q_j$ and its full nullspace to be computed, cost functions should be vector-valued with codomain \mathbb{R}^c , where c reflects the number of constrained degrees-of-freedom. Ideally, the components of the cost function should be independent and follow a common distance metric. Of course, there is no obvious metric to compare distances in translation and rotation, so these components should be weighted such that they contribute equally in the scenarios of interest. Table 5.1 lists cost functions that fulfill these design properties and implement the most relevant types of geometric constraints, including all types of constraints required in the task and motion planning scenarios that will be evaluated in Chapter 6.

To give an example for poor cost function design, consider the cost function Coincident(Point, Line) to be alternatively defined in terms of the distance cost

function $\text{Distance}(\text{Point}, \text{Line})$ with distance zero. However, this alternative function would not reflect the correct number of constrained degrees-of-freedom, which is two in this case. As an effect, the Jacobian matrix would no longer allow us to compute the full nullspace of this constraint.

5.2.3 Completeness of Constrained Sampling

For task and motion planning, it is crucial to ensure that none of the sampling strategies affect the completeness of the search. If only one component of the task planning system failed to cover the complete search space of a given domain, the entire planner would be incomplete and no longer able to solve all scenarios that can be solved, irrespective whether these scenarios are of practical relevance.

For our sampling component to be complete with respect to the search space, it must cover a given constraint manifold. In this context, coverage means that every fully-dimensional open sphere contained in the constraint manifold will be covered by a random sample with a probability larger than zero [118]. The first rigorous proof for the coverage of a sampling approach that projects on constraint manifolds was presented by Berenson and Srinivasa in 2010 [118], as part of proving the probabilistic completeness of the Constrained Bidirectional Rapidly-exploring Random Tree (CBiRRT) path planner. Their proof shows that sampling–projection procedures, including our Algorithm 3, *cover* a given lower-dimensional constraint manifold under several conditions. For their proof to apply to our argumentation, two of its assumptions must be fulfilled [118, p. 3].

First, the projection function, which is the iterative step in Algorithm 3, must update \mathbf{q} if and only if $\mathbf{q} \notin C$. On the one hand, this requires cost functions to have a non-zero derivative outside the constraint space. For example, the point–line distance has a local maximum for points on that line, so its derivative implementation should specifically handle this special case not to converge locally. On the other hand, Algorithm 3 obviously does not step further if a sample $\mathbf{q} \in C$ has been found. Second, if a configuration \mathbf{q} is infinitesimally close to the constraint manifold, it must be projected to the closest point on the manifold. Since the cost is infinitesimal in this case, one can verify that the iterative step is exact and projects to the closest constraint sample.

Even though Berenson and Srinivasa can show the coverage for a wide range of sample–project methods, they also discuss conditions under which these methods fail. Most relevant to us, hierarchical constraints with multiple priorities, which are effective to whole-body motion control [119], cannot be covered by projection sampling. As an example, lower-priority pose optimization should not be integrated

in the constraint sampling routine. Such lower-priority constraints would attract the projection routine to local minima and render the constraint sampling approach incomplete [118, p. 7].

5.2.4 Evaluation of the Sampling Algorithm

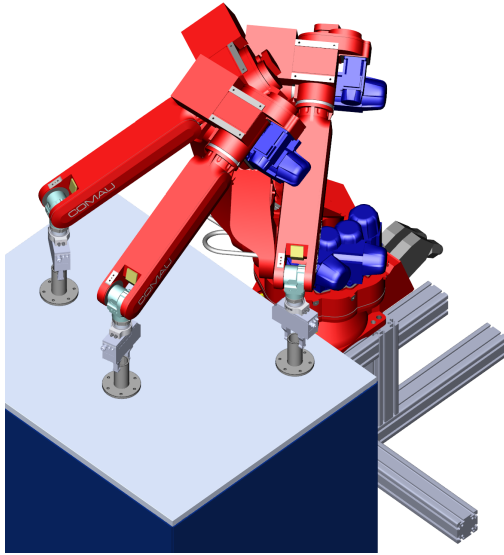
Coverage does not imply that samples are uniformly distributed on the constraint manifold. In order to evaluate our approach to sampling with geometric constraints, we measure the distribution of samples in two relevant scenarios, depicted in Figure 5.4.

In the first scenario, a six-axes manipulator with a parallel gripper is supposed to place an object on a planar table. For this, the sampling algorithm should generate random robot configurations where the tool is at a defined distance to the table, and a certain plane of the object is parallel to the table. To evaluate this scenario, we measure the frequency of samples with respect to their y -coordinate on the table. The distribution, shown in Figure 5.4a, covers the complete range of possible placements in this axis. Although it is clearly denser towards the center in operational coordinates, this may also be explained by the workspace of the manipulator, which offers more configuration-space solutions towards its center.

The second scenario, shown in Figure 5.4b, shows an example where constraint sampling is strongly biased. In this scenario, the same manipulator is supposed to grasp a cylinder at its rim, with one finger inside, one outside. Note that this grasp can be defined by a coincidence, a parallel, and a distance constraint, which all refer to both the tool and the object, but cannot be defined in terms of operational space constraints alone. The constraint space has the form $SO(2)$ and can be parameterized by a grasping angle. The distribution of samples with respect to the grasping angle has a sharp peak and two broader bumps, but it does cover the whole range $]-\pi, \pi]$. Clearly, a large region of unconstrained samples in the configuration space is attracted to the shortest-distance grasp at a certain angle, which causes a peak and a bump around it. The second bump arises from swapping the fingers, which offers fewer solutions. This second example shows that coverage does not imply a uniform distribution of samples even in simple, practical cases.

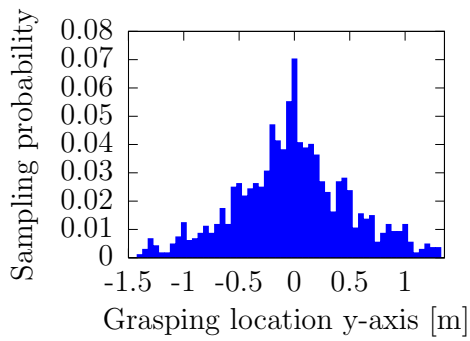
Conclusion

In sum, we propose a formulation for geometric constraints that define relations between robot tool spaces and objects, and provide a projection sampling routine to cover these constraint spaces in robot task and motion planning domains. With this generic sampling method, we can map from abstract, symbolic preconditions of

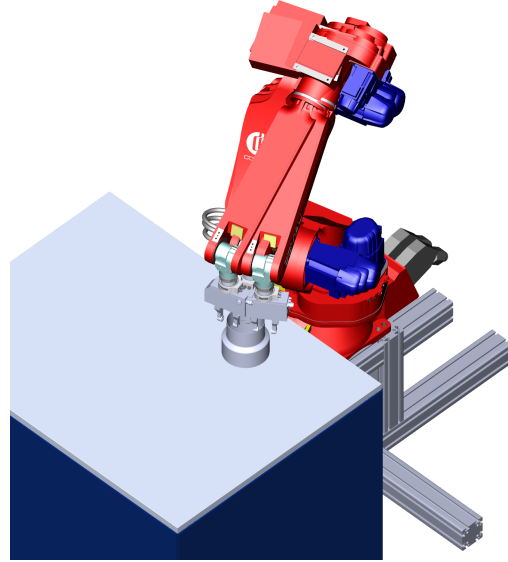


Constraints:

Parallel(Plane tool_Z, Plane table) \wedge
 Distance(Point tool_{center}, Plane table) =
 height(object)

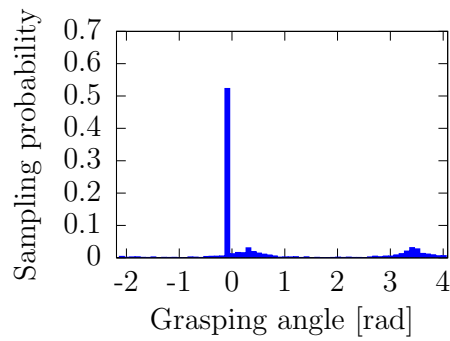


(a) Constraint sample poses for placing an object on a planar table. The distribution is slightly biased toward the center of the workspace.



Constraints:

Parallel(Line tool_Z, Line cylinder_{axis}) \wedge
 Distance(Point tool_{center}, Line cylinder_{axis})
 = radius(cylinder) \wedge
 Coincident(Line tool_X, Plane cylinder_{top})



(b) Constraint sample poses for grasping a cylinder at its rim from a known location. While all grasping angles are sampled, most random poses are attracted to a single grasping location close to the workspace center.

Figure 5.4: Distribution of constraint sampling poses. Even though the constraint space is completely covered, the distribution may be strongly biased, depending on the workspace of the kinematic.

robot actions to feasible geometric states. In particular, the method is suitable to sample in manifolds lower in dimensionality than the full configuration space, where rejection sampling would fail. Compared to related projection sampling methods, the formulation is not limited to the robot tool space, and therefore more expressive with respect to the geometric constraints that can be defined.

Chapter 6

Implementation and Evaluation

The topics discussed in the previous chapters—domain description, automated planning, single-sided geometric predicates, and constrained sampling—form the individual components of our Knowledge-level Action and Bounding Geometry Motion planner (KABouM). In the following, we discuss the implementation of these components and their integration to a software system. Of course, a realistic evaluation can only be conducted by solving different problems on the full system. For this, we define and evaluate different scenarios that cover several robot kinematics, geometries, and symbolic domains. To show the effectiveness of our approach, we further demonstrate individual problem instances on real robot systems.

6.1 System Implementation

An integrated task and motion planner is necessarily a large software system of several components. Besides the symbolic planner, which can be domain-independent, a number of robotics-specific software components need to interact to solve a given problem. While integration of these components is an important design task in such a system, implementation may rely on existing software libraries. In the recent years, powerful robotics software frameworks such as the Robot Operating System [120] have been developed, and open-source robotics software has made substantial progress in general. Our KABouM planner relies heavily on existing software, however, we mostly cherry-pick functions from more lightweight libraries rather than large-scale frameworks. For robotics-specific functions, we reuse many functions of the Robotics Library by Rickert [24]. For collision, inclusion, and distance computations, we use several libraries for comparison [121, 122, 98, 123]. Figure 6.1 shows an overview of the software components in our integrated task and motion planner; components are organized from high-level to low-level in the vertical axis, and the horizontal axis shows the sequence from pre-processing of geometric models to physical execution on a robot system. In the following, we discuss the individual components, in particular the interaction of symbolic and geometric planning.

6.1.1 Geometric Pre-processing

The use of bounding meshes and bounding convex decomposition to approximate geometry is a novel and important feature in the KABouM system. These geometric simplification methods do not require direct integration with the planner itself, but rather run as pre-processing steps before actual planning. In the current version of the implementation, each rigid body of a robot and each object is decomposed with V-HACD [96], using the parameter setting $c = 10^{-4}$, $\alpha = 10^{-3}$ (Section 4.4.1). On

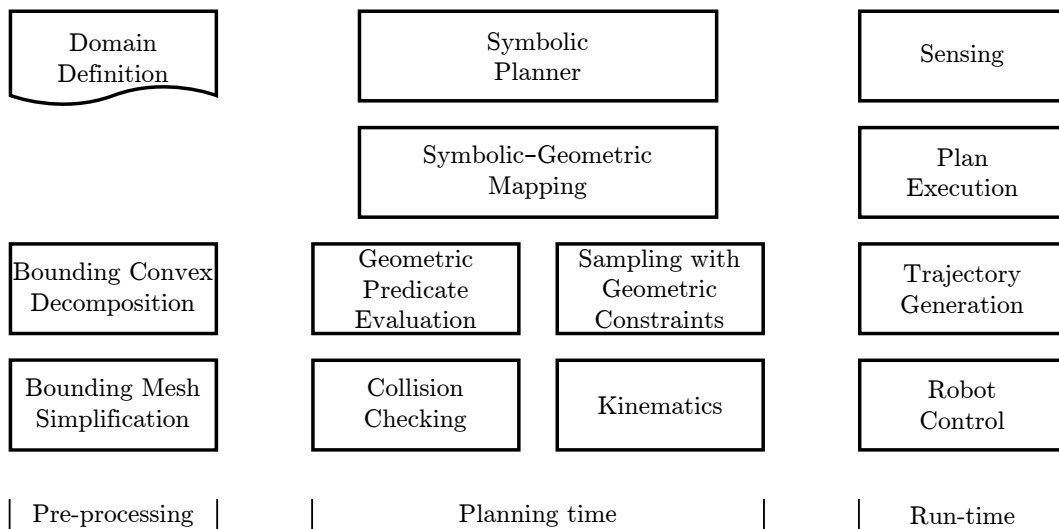


Figure 6.1: Software component overview of the Knowledge-level Action and Bounding Geometry Motion planner (KABouM). The vertical axis loosely reflects software dependencies, with function calls directed from top to bottom. The horizontal axis indicates that pre-processing, planning, and execution are mostly independent and follow a sequence from left to right.

the convex segmentation, the bounding mesh algorithm is run with the cost function E_{planes} with adaption of the constant term (defined in Section 4.3.2) and maximum distance $\varepsilon < 0.02$ m. For the scenarios studied in this chapter, V-HACD requires several minutes of computation time, while the bounding mesh algorithm requires only 30–60 seconds.

6.1.2 Components for Planning and Symbolic–Geometric Mapping

In integrated task and motion planning, the design of a mapping mechanism between symbolic and geometric states is of great importance, as it directly affects the completeness of the search, the expressiveness of the problems that can be formulated, and the performance of the planner to solve complex problems. The simplest way to coordinate symbolic and geometric searches would be to discretize geometry before planning and introduce a symbol for every geometric state. While this method requires no specific interfaces to the symbolic planner, any fixed discretization cannot cover the geometric space and is therefore incomplete, not to mention its extreme branching factor and inefficiency.

Our solution to map between symbolic and geometric states is to define a lexicographical order on geometric states. As a more general alternative, total orders for all components of the geometric state space may be defined. A total order is

sufficient to define a bidirectional map that preserves uniqueness in both directions, between identifiers to geometric states on the symbolic side and full geometric state vectors on the geometric side. Formally, we construct a bidirectional mapping between the geometric state space X and a simpler set of *geometric identifiers* Y . The geometric state space X is usually a product space of robot and object configurations, with several components in \mathbb{R} and $\text{SO}(3)$, but our mapping also allows more general data structures. Depending on the symbolic planner in use, identifiers may be implemented as natural numbers, $Y \subset \mathbb{N}$, as in our case with the PKS planner, or as additional symbols, $Y \subset \Sigma$.

Definition 5 (Symbolic–Geometric Map). *We define a bidirectional mapping function $f : X \mapsto Y$ from the symbolic state space X to a set of identifiers Y using a lexicographical order \preceq (or, other total order) on X . For an existing geometric state $x \in X$, where there exists another state x' with $x \preceq x' \wedge x' \preceq x$, the same identifier $f(x) = f(x')$ is returned. Otherwise, a new identifier y is constructed, for instance $y := |Y| + 1$. The inverse f^{-1} can be implemented efficiently using a storage container with keys Y .*

The bidirectional mapping then has the following properties.

1. (Uniqueness) *Let $x = x'$ denote equality of geometric states $x \preceq x' \wedge x' \preceq x$. Then, $x = x' \iff f(x) = f(x')$.*
2. (Efficiency) *f can be evaluated in time $\mathcal{O}(\log(|Y|))$ using \preceq -sorted trees or other map structures, f^{-1} can be evaluated in constant time using a linear array.*

Implementation of the Symbolic–Geometric Map

As described in Section 3.3, our search scheme is a forward search controlled by the Planning with Knowledge and Sensing (PKS) planner [70]. The symbolic planner calls external procedures and thereby interacts with geometric planning. In particular, the symbolic planner can evaluate geometric predicates, which take symbols and identifiers for geometric states (or, components of the geometric state) as arguments, and return a truth value. For the symbolic planner, a geometric predicate is a function of type $\Sigma \times \Sigma \times \dots \times Y \mapsto \mathbb{B}$. Symbolic–geometric mapping then translates geometric identifiers to states by applying the inverse mapping f^{-1} and calls robotics-specific functions. These functions evaluate whether an action is geometrically feasible, modify the current geometric state, and return true on success. After a successful evaluation, the symbolic planner calls an external procedure of type $\emptyset \mapsto Y$ that returns the identifier of the updated geometric state, generated by

f , and stores that geometric identifier in its symbolic state. Our implementation is motivated by Dornhege's, who refers to the above procedures as condition checking and effect application [5, p. 32].

Note that our approach to symbolic–geometric mapping makes very few assumptions about the symbolic planner, and can therefore interact with most general-purpose planners. Furthermore, it is applicable to all domain-specific state spaces that allow total ordering. When implementing a total order on floating-point values, near equality of imprecise floating-point values must properly be detected and handled to ensure uniqueness.

Mapping Parts of the Geometric State

As a simple variant of this mapping, components of the geometric state space can be mapped individually. With component-wise mapping, we can select exactly those parts of the geometric state on which a geometric predicate depends. Parts of the geometric state that are unrelated to a geometric precondition can be hidden from the symbolic planner. The results of this approach are similar to Dornhege's partial state caching for condition checking [5, p. 103]. As an example, consider a geometric predicate `isReachable(robotSymbol, objectSymbol, objectPositionIdentifier)`, which returns a truth value whether a robot kinematic `robotSymbol` can reach an object `objectSymbol` at a certain position `objectPosition`. When each component of the geometric state space is mapped through a different mapping function, the symbolic identifier `objectPositionIdentifier` only encodes the information needed to evaluate inverse kinematics, the position of the object $f^{-1}(\text{objectPositionIdentifier})$. As a result, the symbolic planner realizes that the geometric predicate is independent from the robot's current configuration and all other objects' positions, and can therefore avoid many future evaluations of that predicate.

On the contrary, in many practical problems, most computation time is spent on collision checking between all robots and all objects. For the problem of collision-free multi-robot manipulation, which embraces all scenarios evaluated in this chapter, most geometric predicates depend on the full state.

Discussion

Given an arbitrary problem instance, it is an important property of the KABouM planner whether or not it will find a solution if one exists; in other words, whether its search scheme is complete. Apart from the internals of the symbolic planner, whose completeness is not in the scope of this discussion, we need to analyze if the search

space is potentially limited by symbolic–geometric mapping, by incomplete sampling, or by geometric predicates that are too strict. Clearly, our bounded ε -precise geometric predicates make the search incomplete. However, the level of incompleteness is controlled by the precision parameter ε . If a solution path exists further than ε from any obstacles (or, a state fulfills inclusion predicates with a penetration depth larger than ε), our bounded ε -precise geometry does not affect completeness. Next, we consider symbolic–geometric mapping. The necessary condition here is that the mapping function f discretizes the full geometric search space to different states of the planner, which is fulfilled by Definition 5. As a result, the symbolic state of the planner, including geometric identifiers Y , describes the full state of the hybrid search.

Finally, we need to discuss whether the interaction of the symbolic planner with geometric predicates, some of which sample at random, will cover the search space. In the external function mechanism described above, new geometric states may be generated by random sampling within a geometric predicate, but a geometric predicate is called only once for each state and each set of arguments. In other words, the geometric branching factor is one. Even for this low branching factor, completeness can always be achieved by introducing a dummy action that increments a random seed and passing that random seed to external functions that need to sample at random. Effectively, the planner would then add edges in the search graph that only modify the random seed, and sample the full geometric space while progressing deeper.

In practical applications of robot manipulation planning, coverage of the geometric search space can also be achieved by a combination of a random robot motion action with constrained-space sampling actions pick and place. Since a robot motion action can be applied in every state, it covers the space of robot configurations. Constrained-space sampling actions should then take the current geometric configuration as a starting point, and cover the full space of robot and object configurations as an effect.

To sum up, symbolic–geometric mapping does not limit completeness in the types of scenarios studied in this chapter. Assuming that the symbolic planner covers its state space, we only limit the search space by single-sided ε -precise approximation of the geometry, which is fully intentional.

Robotics Components at Planning Time

When the symbolic planner calls an external function, its arguments are mapped to geometric states and evaluated by robotics-specific functions. These functions

perform kinematic and geometric checks, generate random configurations or configurations fulfilling geometric constraints, and generate paths. Robotics components make use of forward and inverse kinematics, sampling with geometric constraints, evaluation of all bounded geometric predicates with collision and inclusion checking, and generation of paths in joint and Cartesian spaces.

At the action level, an external function for each type of symbolic action is implemented. While standard action implementations for robot transit and transfer motion are available, more scenario-specific actions require implementation of a callback function, which constructs a path and calls geometric predicates. By re-using utility functions, scenario-specific functions can be kept short. As an example, the implementation of a dual-arm handover action needs to construct a Cartesian path, but it can generate its waypoints using constraint sampling, propagate object motion using utility functions, and check for collisions using geometric predicates. All lower-level robotics functions for robot control, kinematics, collision and inclusion checking, distance queries, trajectory interpolation, and visualization rely on the Robotics Library [24]. Collision, inclusion, and distance queries are internally handled by the Bullet [121], Solid [122, 124], and ODE [98] libraries. For the scenarios studied in this chapter, we derived and implemented an efficient closed-form solution for inverse kinematics (Appendix A.3).

6.1.3 Run-time Components

After a plan is solved with its symbolic actions and robot paths, it is passed to an execution component. For execution, the plan is traversed from its root node. In case of a branched plan (Section 3.3.3), the execution component triggers a runtime sensing action at each branch. A sensing component retrieves the result from hardware sensors and returns a binary result, based on which execution selects a positive or negative branch in the plan. All other actions of the plan consist of a sequence of robot waypoints and other hardware events, such as opening or closing a gripper. In order to generate smooth trajectories, robot paths are interpolated by quintic polynomials. Finally, trajectories are sampled at the control frequency of the robots and joint angles are sent to hardware robot controllers.

6.2 System Evaluation

In this section, we evaluate our KABouM planner on several scenarios, which illustrate the different features of the integrated task and motion planning system. In contrast to the FORCE SENSING scenario discussed in Section 3.3.3, all scenarios

described in this chapter are solved in an integrated search space of symbolic and geometric states. Computation times are benchmarked on a desktop computer with a dual-core 2.8 GHz processor and 12 GB of random access memory. A number of problem instances are additionally demonstrated in real robot setups.

First, we study pick-and-place tasks in a bimanual setup of two industrial manipulators: the BARTENDER scenario, which we demonstrated in an earlier work [6], and a slightly more intricate BIMANUAL CIRCULAR REARRANGE scenario, where goal positions collide with other objects' initial positions. As a second type of scenario, we discuss the STACKED n OBJECTS scenario, where a single mobile manipulator solves a "blocks domain" problem that is combinatorially challenging. Finally, we study the industrial manufacturing scenario BIMANUAL ASSEMBLY, where two manipulators assemble a gearbox component. Contrary to the earlier pick-and-place scenarios, BIMANUAL ASSEMBLY involves more complex types of assembly actions, some of which require bimanual manipulation.

6.2.1 Bimanual Pick-and-Place Scenarios

To illustrate the solution of bimanual pick-and-place tasks, we describe two closely related scenarios. Both scenarios contain two six-degree-of-freedom robots with compliant humanoid hands, and several bottles that are initially located on a table. Because of the width of the table, most objects and locations are only reachable by one of the robots. Robots can pick and place objects, and move to random waypoints. The formal, symbolic domain definition of both scenarios is listed in Table 6.1.

In the BARTENDER scenario, the goal is to place all empty bottles inside a certain area, named *dishwasher*, which is only reachable by robot₂. In order to learn whether a bottle is empty or not, a sensing action `senseSelfEmpty` is available. Only at runtime, the system will receive the sensing result from a visual object recognition component. Therefore, a branched plan with sensing actions must be generated. In the BIMANUAL CIRCULAR REARRANGE scenario, the goal is to place all objects at the initial location of another object, following a circular order. Of course, this is an intricate rearrangement task, because all objects act as obstacles and obstruct to goal location of another.

Discussion of the Bartender Scenario

In contrast to the FORCE SENSING scenario discussed in Section 3.3.3, the visual sensing action `senseSelfEmpty` in the BARTENDER scenario has no preconditions. Therefore, the planner decides to execute all `senseSelfEmpty` actions first, and then

Table 6.1: Symbolic domain definition \mathcal{S} of the BARTENDER and BIMANUAL CIRCULAR REARRANGE scenarios.

Action	Preconditions	Effects
move(robot r)	true	(no symbolic effects)
pickUp(robot r , object o)	$K(\text{isHandEmpty}(r)) \wedge$ $K(\neg\text{isGrasped}(o))$	add(\mathcal{K}_f , $\neg\text{isHandEmpty}(r)$), add(\mathcal{K}_f , $\text{isGrasped}(o)$), add(\mathcal{K}_f , $\text{isRobotGrasps}(r, o)$)
putDown(robot r , object o , location l)	$K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\text{isGrasped}(o))$	add(\mathcal{K}_f , $\text{isHandEmpty}(r)$), add(\mathcal{K}_f , $\neg\text{isGrasped}(o)$), add(\mathcal{K}_f , $\text{isAtLocation}(o, l)$), del(\mathcal{K}_f , $\text{isRobotGrasps}(r, o)$)
BARTENDER: senseIfEmpty (object o)	true	add(\mathcal{K}_w , $\text{isEmptyBottle}(o)$)

Domain Element	Element Definition
Types	object, robot, location
Constants	object o_1, o_2, \dots, o_n robot $\text{robot}_1, \text{robot}_2$ BARTENDER: location any, dishwasher BIMANUAL CIRCULAR REARRANGE: location any, $\text{loc}_1,$ $\text{loc}_2, \dots, \text{loc}_n$
Predicates	isGrasped, isRobotGrasps, isHandEmpty, isAtLocation BARTENDER: isEmptyBottle
Initial knowledge \mathcal{K}_f	isHandEmpty(robot_1), isHandEmpty(robot_2), $\neg\text{isGrasped}(o_1), \neg\text{isGrasped}(o_2), \dots, \neg\text{isGrasped}(o_n)$
Goal criteria G	BARTENDER: forall(object o)($K(\text{isAtLocation}(o,$ dishwasher)) $\wedge K(\neg\text{isGrasped}(o)) \oplus$ $K(\neg\text{isEmptyBottle}(o))$) BIMANUAL CIRCULAR REARRANGE: $K(\text{isAtLocation}(o_1,$ $\text{loc}_2)) \wedge K(\text{isAtLocation}(o_2, \text{loc}_3)) \wedge \dots \wedge$ $K(\text{isAtLocation}(o_n, \text{loc}_1))$

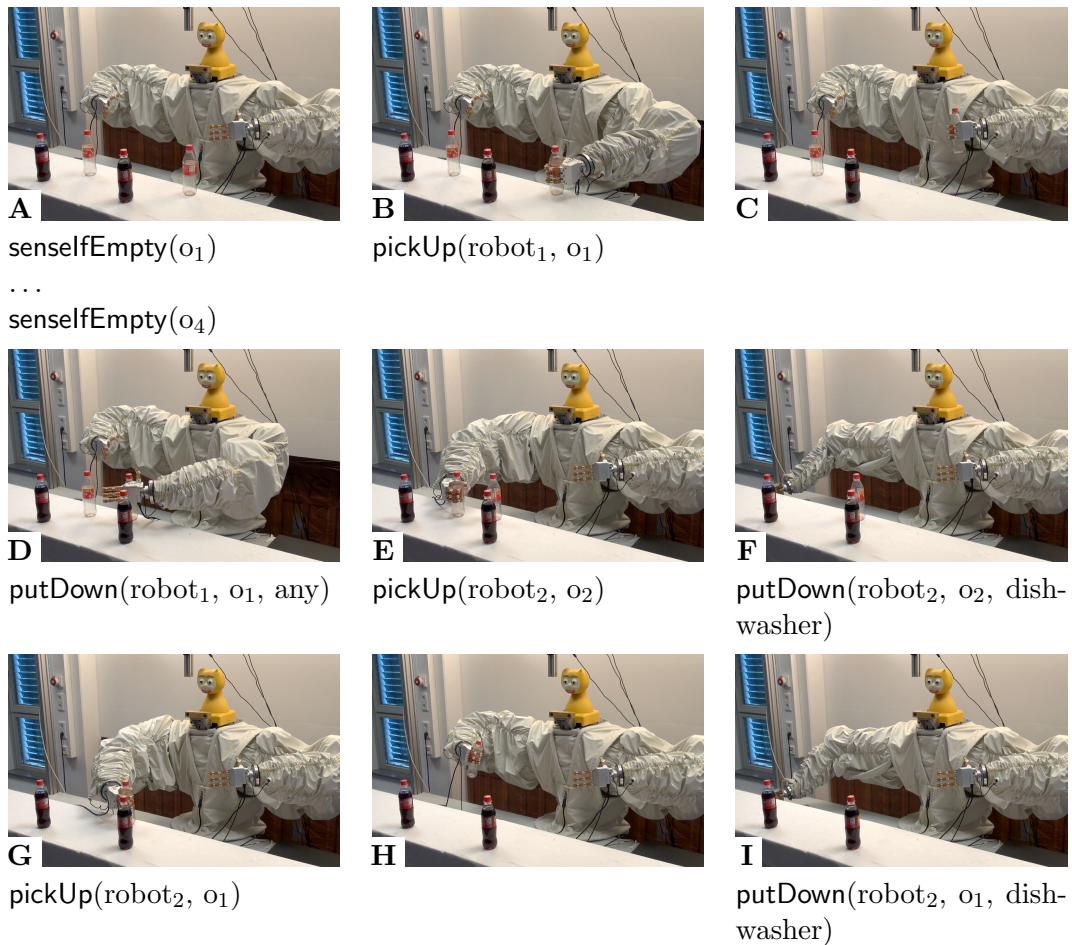


Figure 6.2: Action sequence of a solution in the BARTENDER scenario [6]. In order to transfer o_1 to a dishwasher location, robot_1 has to place it at a location where the workspaces of both robots intersect (Image **D**).

generates pick-and-place actions for each possible outcome. An action sequence for an instance of four bottles, two of them empty, is shown in Figure 6.2. This sequence was solved and executed by an early version of our system, details of which are described in [6]. In this scenario, we can observe that the KABouM planner automatically solves *sequences of transfers to overcome workspace limitations* of multiple robots. In this example, object o_1 can only be reached by robot_1 , which in turn cannot reach the goal location dishwasher. As a viable plan, o_1 is placed at a location in reach of both kinematics (Figure 6.2 **D**), and robot_2 can finally transfer it to a goal location.

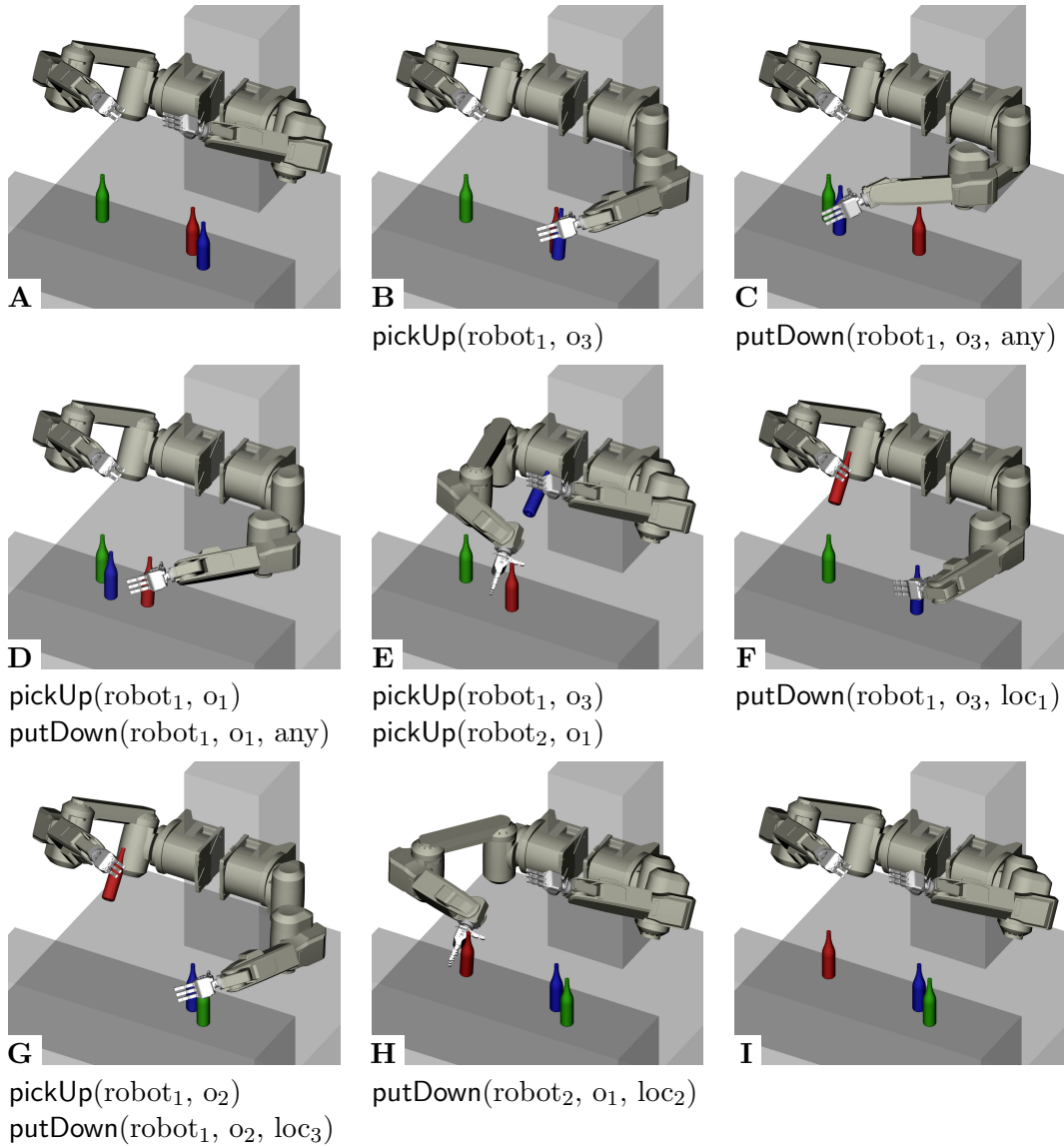


Figure 6.3: Action sequence of the shortest solution for a BIMANUAL CIRCULAR REARRANGE instance of three objects that are to be reordered. In a circular rearrange, objects block the goal location of another. In this scenario, some areas are only reachable by one of the two manipulators, and grasps are likely to collide with other objects.

Table 6.2: Evaluation of the BIMANUAL CIRCULAR REARRANGE scenario with three objects. Computation time and procedure calls are measured with respect to the search strategy of the symbolic planner. Averages are taken over 10 trials, standard deviation is shown in gray.

	Depth-first Search	Breadth-first Search	Iterative Deep- ening Search
Total Time [s]	0.4119 ±0.0055	4.8699 ±0.7699	17.4939 ±5.7788
Symbolic Planning [s]	0.0518 ±0.0007	0.5905 ±0.1092	2.2185 ±0.7653
Geometric Search [s]	0.3601 ±0.0048	4.2794 ±0.6607	15.2754 ±5.0134
Inverse Kinematics Calls	13969 ±229	116722 ±15268	409582 ±148753
Inverse Kinematics [s]	0.0210 ±0.0005	0.1958 ±0.0283	0.6860 ±0.2429
Collision Checks	2197 ±16	26229 ±4366	93624 ±30699
Collision Checking [s]	0.2913 ±0.0010	3.5438 ±0.5422	12.6297 ±4.1337
Positive Collision Checks [%]	15.94 ±2.65	20.84 ±0.56	17.68 ±1.23
Geometric States	412 ±14	2067 ±388	2525 ±52
Actions in Solved Plan	440.00 ±7.07	10.66 ±0.47	22.00 ±2.82

Discussion of the Bimanual Circular Rearrange Scenario

Compared to the BARTENDER scenario, transfer actions in the BIMANUAL CIRCULAR REARRANGE scenario are more likely to collide with other objects, because objects' initial and goal locations are identical up to a circular rearrangement. A typical solution for an instance with three objects is shown in Figure 6.3. In this plan, many of the possible object-object collisions are avoided by picking up two objects with both robots (Figure 6.3 E).

To evaluate this scenario, we measure computation times of various components of the planner and compare the three search schemes depth-first, breadth-first, and iterative deepening search. The benchmark results for an instance with three objects are listed in Table 6.2. In general, geometric search requires most computation time, with most time spent on collision checking of robot paths. Depth-first search is fastest at solving this scenario, and constructs the smallest number of distinct geometric states. However, its plans are too long to be executed in a real setup. Note that even breadth-first search, which produces rather short plans, is not guaranteed to find the shortest possible plans because of the limited geometric branching factor.

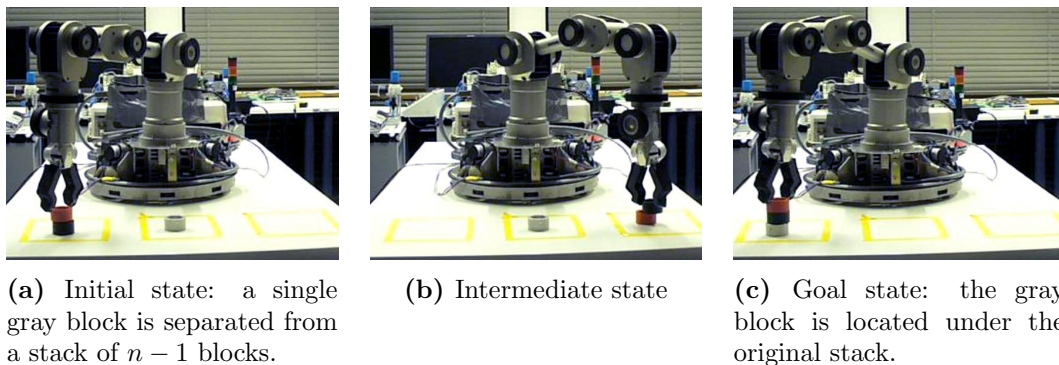


Figure 6.4: The STACKED n OBJECTS scenario was implemented and demonstrated on a mobile manipulator with $n = 3$ by S. Nogina [125]. (Photographs provided by S. Nogina)

6.2.2 Stacked n Objects Scenario

In the task and motion planning scenarios discussed so far, collisions are common and prevent many actions, but already small changes to the plan made most actions feasible. As in many pick-and-place scenarios, an object that presents an obstacle and would interfere with too many actions can usually be parked at a location in free space that does not interfere. In the STACKED n OBJECTS scenario, we purposely define a more difficult environment and goal criterion to explore the limits of our KABouM planner. For this, we drastically reduce the free space, and choose a symbolic domain and goal criterion that cannot be decomposed into independent subgoals. In this scenario, a mobile manipulator is supposed to move a new block under an existing stack of $n - 1$ blocks, keeping the order of the existing stack. Only three locations are available to stack blocks. Blocks may only be picked up from and put down to the top of a stack, or put down at a free location to form a stack of a single block. Note that this scenario was already discussed in a 2013 master thesis [125] and another earlier work [9], but is here evaluated with a newer and substantially different version of the planning system. Figure 6.4 shows this scenario for the case $n = 3$, where the initial stack is located at the left location, and the new, gray block at the center. After several pick and place actions, the gray block is placed under the original stack, which is the only placement to fulfill the goal criterion.

Both the symbolic goal and the small number of free locations make this scenario particularly hard to solve. First, only three stacks are available to place objects. Therefore, free space is very limited and objects that prohibit other actions cannot simply be “parked” at a different location. As a result, the search for feasible object transfers is a combinatorial one rather than a geometric one. Second, the symbolic

Table 6.3: Symbolic domain definition \mathcal{S} of the STACKED n OBJECTS scenario.

Action	Preconditions	Effects
move(object this, object other)	$K(\text{isHighest}(\text{this})) \wedge$ $K(\text{isHighest}(\text{other}))$ \wedge $\neg K(\text{isAStack}(\text{this}))$	$\text{del}(\mathcal{K}_f, \text{isHighest}(\text{other})),$ $\text{forall}(\text{object } o) (K(\text{getOneBelow}(\text{this}) =$ $o) \implies \text{add}(\mathcal{K}_f, \text{isHighest}(o))),$ $\text{add}(\mathcal{K}_f, \text{getOneBelow}(\text{this}) = \text{other})$
Domain Element	Element Definition	
Types	object	
Constants	object $o_1, o_2, \dots, o_n, \text{stack}_1, \text{stack}_2, \text{stack}_3$	
Predicates	isAStack, isHighest	
Functions	getOneBelow : object \mapsto object	
Initial knowledge \mathcal{K}_f	isAStack(stack ₁), isAStack(stack ₂), isAStack(stack ₃), isHighest(o_n), isHighest(o_1), isHighest(stack ₃), getOneBelow(o_n) = o_{n-1} , getOneBelow(o_{n-1}) = o_{n-2} , \dots , getOneBelow(o_3) = o_2 , getOneBelow(o_2) = stack ₁ , getOneBelow(o_1) = stack ₂	
Goal criteria G	$K(\text{stack}_1 = \text{getOneBelow}(o_1)) \wedge K(o_1 =$ $\text{getOneBelow}(o_2)) \wedge K(o_2 = \text{getOneBelow}(o_3))$ $\wedge \dots \wedge K(o_{n-1} = \text{getOneBelow}(o_n))$	

task is particularly hard because subgoals are highly dependent and do not allow decomposition [15, p. 378]. States that fulfill part of the goal criterion are usually many steps away from the single goal state. Note that the initial state already meets almost all conjunctions of the goal criterion, even though it is at least $2n - 1$ symbolic actions away from the goal state. In this way, the STACKED n OBJECTS scenario clearly shows Sussman's anomaly [126], in that decomposition approaches fail and a global search is required to find a valid plan. One can observe that choices early in plan, such as blocking a single object by stacking an object on top of it, may render the goal infeasible and entails a long backtracking search after many actions have been evaluated. Importantly, the benefit of an action cannot be rated by standard heuristics. In particular, partial fulfillment of the goal conjunction is not a helpful metric. At the same time, correct plans are sparse in the search space. The combination of these properties makes this scenario difficult, such that valid solutions can only be found in an exhaustive, global search.

Domain Definition

The symbolic domain for this scenario is defined in Table 6.3. Since the geometric state space is discrete, the preconditions and effects of transferring objects between stacks can be modeled symbolically. One way to define this problem is to introduce symbols for all stack locations and objects, and order these by a mapping `getOneBelow`, which maps from one object to the object or stack located directly below. Rather than describing this relation by a binary predicate, this function notation allows a simpler formulation of the effects of the transfer action `move`. For most symbolic planners other than the PKS planner with its function mechanism, a more verbose formulation with a binary predicate could replace this syntax. The central part of the symbolic domain definition is the single action `move`. Using this action, the robot picks up an object `this` and puts it down on top of an object or stack location `other`. Clearly, both objects involved must be on top of a stack, which is available through a predicate `isHighest`. As an effect of this action, the second object or stack location is no longer the top of a stack. Also, the function value `getOneBelow(this)` for the transferred object is updated to represent the target location. Finally, the object or stack location that was previously located under the transferred object becomes top of a stack. In the current implementation of the symbolic planner, this latter expression requires iteration over all objects with the `forall` syntax.

At the geometric level, the STACKED n OBJECTS scenario requires only generation of grasp poses and collision checking for all motion paths. Since the robot only needs to manipulate objects on top of a stack, the default grasping angle always succeeds, and swept volumes of all robot paths are always free of collisions for the sizes of stacks covered by our evaluation.

Example Solution

Figure 6.5 shows a solution for the STACKED n OBJECTS with $n = 4$ objects. Since the solution was found through a breadth-first search, it is shortest in terms of the number of symbolic actions. Essentially, the robot first relocates the stack of $n - 1$ objects to a free location (Figure 6.5 **A–F**). It can then move the gray block o_1 to its goal location (Figure 6.5 **G**), and finally stack all other blocks as required (Figure 6.5 **H–L**). Note that, even though most blocks are depicted as blue, they are not interchangeable and the goal criteria require the final stack to be ordered from o_1 the lowest to o_n on top.

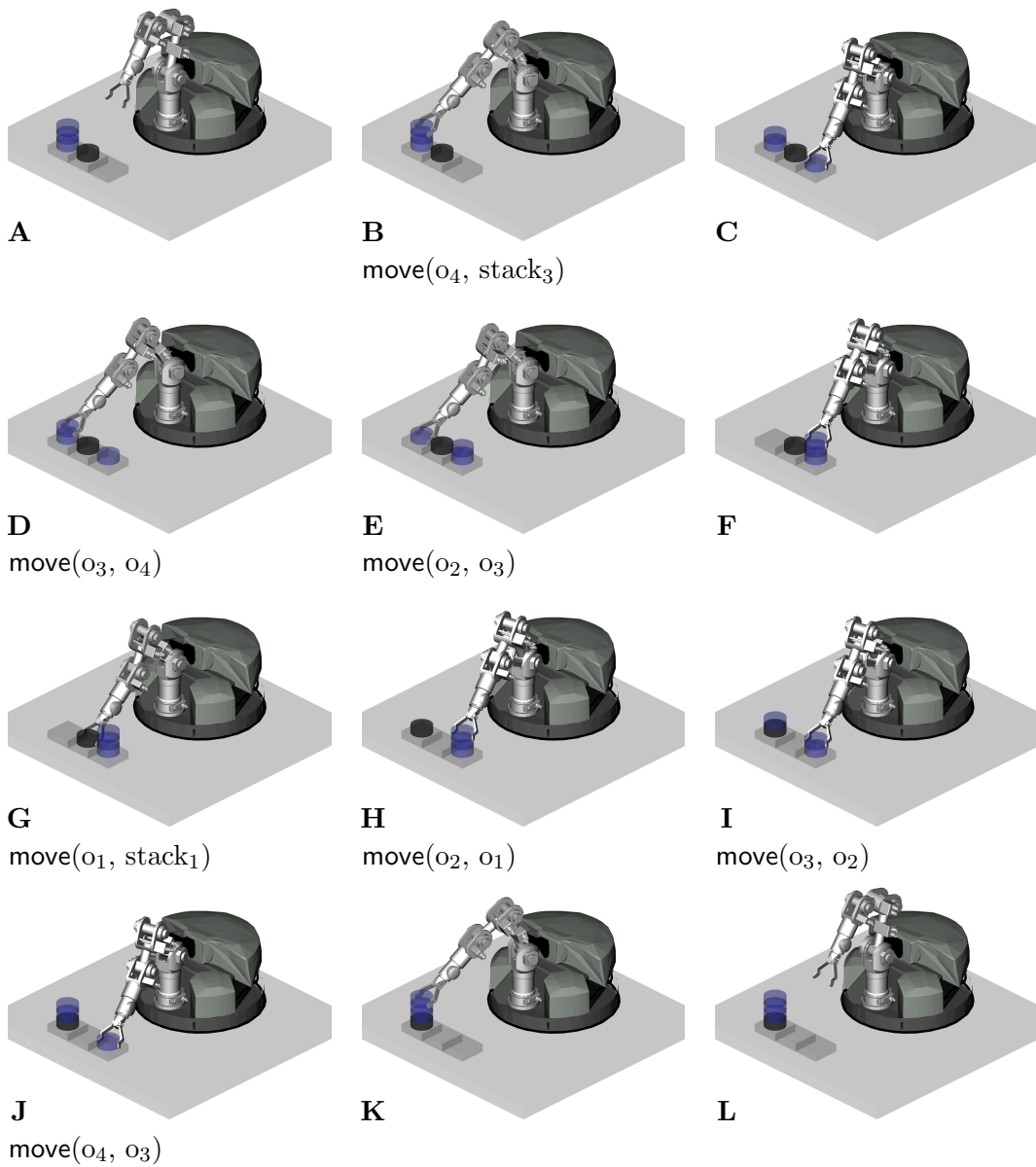


Figure 6.5: Action sequence of the shortest solution for the STACKED n OBJECTS scenario with $n = 4$ objects. The robot is supposed to place the gray block under an existing stack of three blocks (shown in blue), and may only move objects on top of three stacking locations. The solution was obtained by a breadth-first search, requiring 50 ms for symbolic planning and 17 ms for geometric planning, including inverse kinematics and collision checking.

Table 6.4: Evaluation of the STACKED n OBJECTS scenario, where n objects are to be stacked in a certain order, using only three stacks. Computation time and procedure calls are measured with respect to the number of objects n in the problem instance and the search strategy of the symbolic planner.

Number of objects n	Total Time [s]	Symbolic Planning [s]	Geometric Search [s]	Inverse Kinematics		Collision Checking		Symbolic Actions in Solution
				Calls	Time [s]	Calls	Time [s]	
Depth-first symbolic search								
2	0.017	0.001	0.015	14	0.00003	158	0.015	7
3	0.055	0.005	0.050	94	0.00011	752	0.049	47
4	0.329	0.041	0.287	556	0.00060	4449	0.281	278
5	2.224	0.430	1.792	3096	0.00349	27796	1.758	1548
6	22.787	4.175	18.610	28058	0.03391	287931	18.286	14029
7	286.064	86.302	199.550	266062	0.37553	3089850	196.133	133031
8	(timeout after 300 seconds)							
Breadth-first symbolic search								
2	0.009	0.001	0.008	6	0.00001	139	0.008	3
3	0.018	0.005	0.012	10	0.00001	192	0.012	5
4	0.067	0.050	0.017	14	0.00001	263	0.016	7
5	0.477	0.454	0.022	18	0.00002	355	0.021	9
6	4.658	4.627	0.030	22	0.00003	474	0.030	11
7	(timeout after 300 seconds)							
Iterative deepening symbolic search								
2	0.011	0.001	0.009	10	0.00001	141	0.009	5
3	0.025	0.012	0.013	14	0.00001	194	0.012	7
4	0.329	0.302	0.026	32	0.00004	408	0.025	16
5	0.744	0.718	0.025	28	0.00003	396	0.025	14
6	7.204	7.166	0.038	36	0.00004	588	0.037	18
7	112.615	110.243	1.618	50	0.04056	927	1.462	25
8	(timeout after 300 seconds)							

Quantitative Evaluation

Since the STACKED n OBJECTS scenario allows an automatic generation of domain definitions controlled by the number of objects n , we can study our KABouM planner more quantitatively and evaluate its behavior and performance with respect to the problem size n . In addition to the parameter n , we again measure its performance under three different search strategies for symbolic planning—depth-first search, breadth-first search, and iterative deepening. Since this scenario is fully deterministic, a single measurement suffices for each instance. The results are listed in Table 6.4.

The most important observation is that the total planning time, while short for small numbers of objects, increases tremendously with each additional object. After seven objects, the problem becomes computationally infeasible. Apparently, the symbolic search time grows super-polynomially with the size of the problem. For $n = 8$ objects and for all three search schemes, symbolic planning exceeds all available random-access memory, after which a successful solution can no longer be expected. Since geometric planning is only performed once for each move, and inverse kinematics and collision checking succeed at the first try in the scenario, the number of geometric queries scales almost proportionally with the number of symbolic actions. Search schemes that produce short symbolic solutions, breadth-first and iterative deepening searches, therefore require very little geometric planning. In the $n = 7$ instances, computation time was already prolonged by some memory swapping, which also affected the efficiency of geometric computations. In all smaller instances, geometric search time is proportional to the number of symbolic actions in a solution.

In general, the two types of geometric queries contribute only little to the overall search time. Since the grasp planning and inverse kinematics routine was manually derived and implemented for the five degree-of-freedom rotational manipulator, it requires very little computation time. Collision queries are slightly higher in numbers because multiple queries are needed to ensure the absence of obstacles along the path of a transfer motion. Note that a linear configuration-space path for transferring a block from one stack to another does not collide in this scenario, which keeps motion planning very simple. With bounding sets of convex polyhedra available for all objects and the robot manipulator, collision checks are purely convex and can be computed with the Gilbert/Johnson/Keerthi (GJK) algorithm [100]. Using the implementation within the Solid library [124], an average collision check requires no more than $100 \mu\text{s}$.

To summarize, we voluntarily included the STACKED n OBJECTS scenario in

order to explore the computational limitations of the Knowledge-level Action and Bounding Geometry Motion planner. Even though the symbolic domain definition of this scenario is not very complicated, more domain-specific knowledge would be required to solve larger problem instances. This result shows the importance of future work in symbolic planning and motivates further research in heuristic search schemes. Future enhancements to the heuristic search of the PKS planner will directly improve the results of the integrated system.

6.2.3 Bimanual Assembly Scenario

While the previous scenarios highlighted individual aspects of the KABouM planner, *BIMANUAL ASSEMBLY* is a more complex scenario with several types of objects and actions. In this scenario, two industrial manipulators are supposed to manufacture a component of a gearbox (Figure 6.6f). The two robots are equipped with two kinds of parallel grippers, capable of grasping objects at different poses and assembling different parts. All four objects necessary for the final product are initially located on a table, depicted in Figure 6.7a. For a valid goal state in this scenario, a gearbox object must be located on the table (Figures 6.7b, 6.6f). The *BIMANUAL ASSEMBLY* scenario has been demonstrated with a novel, interactive robot programming system [127] as part of an industrial use-case of the *SMErobotics* project [128], but has not been solved with integrated task and motion planning before.

The initial types of available objects are a bearing, a pipe, and a mechanical tree (Figure 6.6a–c). A bearing can be grasped by both robots with different tools; one robot can grasp a bearing from its inner side, the other robot from outside. The inner grasp is necessary to insert a bearing into a pipe, which forms a subassembly (Figure 6.6d); because tolerances are low, both parts must be held by robots and assembled bimanually. The outer grasp allows assembling a bearing and a mechanical tree (Figure 6.6e). In addition, each robot can hand over a bearing to the other one. To complete the set of possible actions, robots can move to a different configuration, which also allows transferring grasped objects, and both subassemblies can be assembled to create a gearbox component. Visual examples for all these actions are shown in Figure 6.8.

Domain Definition

In order to model the assembly actions in this scenario symbolically, we introduce unary predicates to represent the type of an object, *isABearing*, *isAPipe*, *isATree*, and *isAGearbox*. An assembly action then requires objects of specific types as a precondition, and changes the type of an object as an effect. For the other object

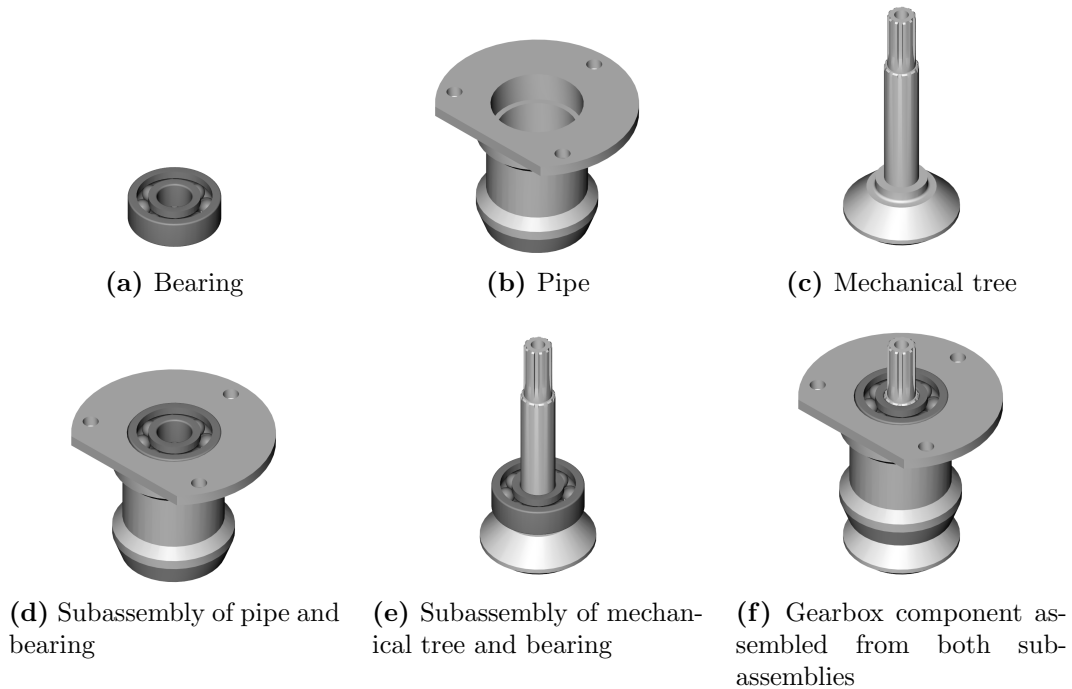


Figure 6.6: Objects in the BIMANUAL ASSEMBLY scenario. The final gearbox is assembled from two bearing objects, a pipe object, and a mechanical tree object.

that is being assembled, the `isContained` predicate is set, so it can no longer be manipulated individually. The full symbolic action definition is listed in Table 6.6. To complete the symbolic domain definition, pick-and-place behavior with multiple robots is modeled by the predicates `isGrasped`, `isRobotGrasps`, and `isHandEmpty`, similar to the BARTENDER scenario described earlier. All predicates and a problem instance with the minimum set of objects are given in Table 6.5.

In addition to the symbolic preconditions listed in Table 6.6, every symbolic action in this domain includes exactly one geometric function as a precondition. If all symbolic preconditions are fulfilled, an action is evaluated geometrically by passing its arguments and the current geometric state identifier. Action `move` samples a new robot configuration at random, and all other actions generate their waypoints relative to the current configuration, most of them by solving geometric constraints. After that, the newly generated path is checked for collisions. If any of these steps fail, `false` is returned to the symbolic planner. On success, the geometric state is propagated, and the symbolic planner receives its updated identifier by calling an external function as part of the action's effects.

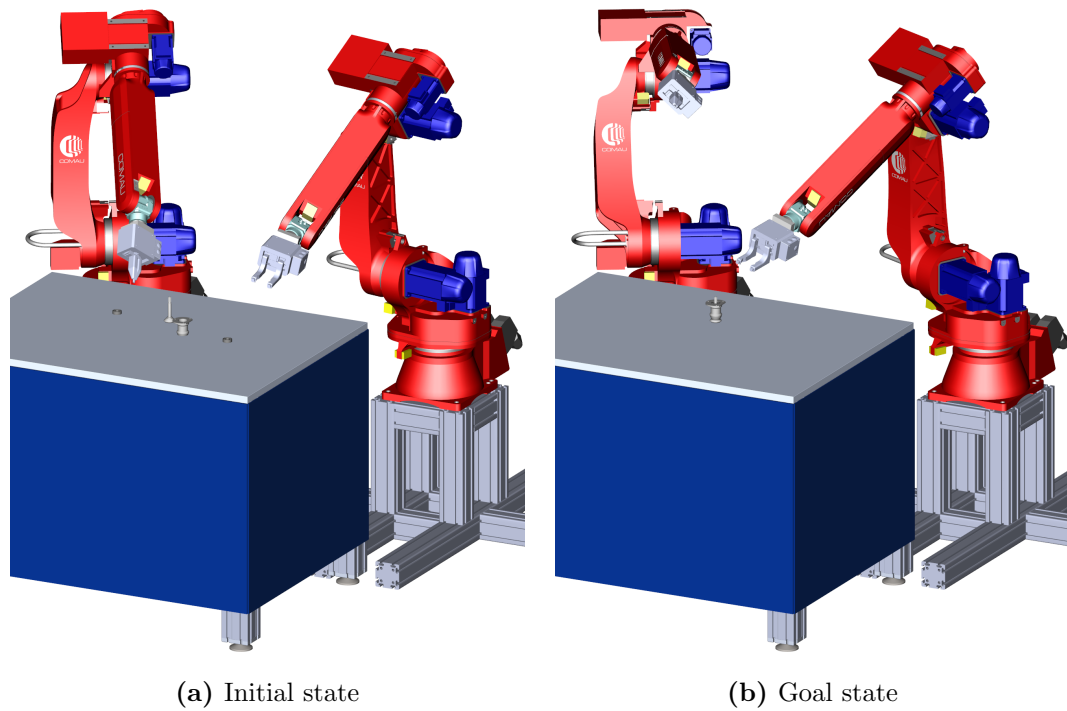


Figure 6.7: Initial and goal states in an example solution of the BIMANUAL ASSEMBLY scenario.

Evaluation

For evaluation, we again measure computation time for the several steps of the KABouM planner, compare between the three search schemes, and observe the difference between bounded geometric predicates and an unoptimized geometry. The results are listed in Table 6.7.

In general, all search schemes succeed to create viable plans under all geometric representations and collision checking libraries. We first analyze the difference in performance between our bounding convex decomposition of the geometry compared to conventional triangle meshes. When using a bounding convex decomposition of the geometry, it takes less than one second to solve the BIMANUAL ASSEMBLY scenario in a depth-first search, and less than twenty seconds in other search schemes. In contrast, in a conventional geometry of triangle meshes, KABouM needs more than ten times the computation time, under all search schemes.

Because the performance of a collision library could easily be affected by sub-optimal settings or poor choice of broad-phase algorithms, we take additional measurements on two completely different collision libraries. The Solid collision library [122] only implements convex-convex checking, which is used by our bounded geometric

Table 6.5: 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 \mathcal{K}_f	isABearing(o_1), isABearing(o_2), isAPipe(o_3), isATree(o_4), isHandEmpty($robot_1$), isHandEmpty($robot_2$), \neg isContained(o_1), \neg isContained(o_2), \neg isContained(o_3), \neg isContained(o_4), \neg isGrasped(o_1), \neg isGrasped(o_2), \neg isGrasped(o_3), \neg isGrasped(o_4)
Goal criteria G	exists(object o) (K (isAGearbox(o)) \wedge K (\neg isGrasped(o)))

predicates. Its performance to evaluate bounded geometric predicates is equal to or slightly better than the Bullet physics library. The Open Dynamics Engine [98] in turn provides collision checking for triangle meshes. Compared to Bullet, its computation time is approximately equal. As a result, bounded geometric predicates are significantly more efficient in this scenario, irrespective of a particular collision checking library. Based on these measurements, we formulate the following result.

Proposition 4. *Bounded geometric predicates, which operate on a bounding convex decomposition of the exact geometry, allow more efficient robot task and motion planning compared to geometric predicates that operate on triangle meshes.*

After comparing different types of geometry and collision libraries, we consider the different search schemes. On average, depth-first search is faster to find a solution, but its solutions contain more actions than needed (of all types). While breadth-first search and iterative deepening search also evaluate all types of actions, their final plan is of minimal length, or contains one additional move action. During planning, most time is spent on geometric evaluation, in particular collision checking of robot paths. Even though all types of collisions occur—robot/object, robot self-collisions, object/object, robot/robot—only few generated paths must be discarded because of collisions.

The search schemes show different behaviors of how many distinct geometric states are generated. Breadth-first search builds a graph of hundreds of distinct

Table 6.6: 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)	$K(\text{isHandEmpty}(r)) \wedge$ $K(\neg \text{isGrasped}(o)) \wedge$ $K(\neg \text{isContained}(o))$	$\text{add}(\mathcal{K}_f, \neg \text{isHandEmpty}(r)),$ $\text{add}(\mathcal{K}_f, \text{isGrasped}(o)),$ $\text{add}(\mathcal{K}_f, \text{isRobotGrasps}(r, o))$
putDown(robot r, object o)	$K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\text{isGrasped}(o)) \wedge$ $K(\neg \text{isContained}(o))$	$\text{add}(\mathcal{K}_f, \text{isHandEmpty}(r)),$ $\text{add}(\mathcal{K}_f, \neg \text{isGrasped}(o)),$ $\text{del}(\mathcal{K}_f, \text{isRobotGrasps}(r, o))$
assemble PipeBearing(robot r, object o, robot r', object o')	$K(r \neq r') \wedge$ $K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\text{isRobotGrasps}(r', o')) \wedge$ $K(\text{isAPipe}(o)) \wedge$ $K(\text{isABearing}(o')) \wedge$ $K(\neg \text{isContained}(o)) \wedge$ $K(\neg \text{isContained}(o'))$	$\text{add}(\mathcal{K}_f, \text{isHandEmpty}(r')),$ $\text{add}(\mathcal{K}_f, \text{isContained}(o')),$ $\text{add}(\mathcal{K}_f, \text{containsBearing}(o)),$ $\text{del}(\mathcal{K}_f, \text{isRobotGrasps}(r',$ $o'))$
handoverBearing (robot r, object o, robot r')	$K(r \neq r') \wedge$ $K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\text{isABearing}(o)) \wedge$ $K(\neg \text{isContained}(o)) \wedge$ $K(\text{isHandEmpty}(r'))$	$\text{del}(\mathcal{K}_f, \text{isRobotGrasps}(r, o)),$ $\text{add}(\mathcal{K}_f, \text{isRobotGrasps}(r',$ $o)),$ $\text{add}(\mathcal{K}_f, \text{isHandEmpty}(r)),$ $\text{add}(\mathcal{K}_f, \neg \text{isHandEmpty}(r'))$
assemble BearingTree(robot r, object o, object o')	$K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\neg \text{isGrasped}(o')) \wedge$ $K(\text{isABearing}(o)) \wedge$ $K(\text{isATree}(o')) \wedge$ $K(\neg \text{isContained}(o)) \wedge$ $K(\neg \text{isContained}(o'))$	$\text{del}(\mathcal{K}_f, \text{isRobotGrasps}(r, o)),$ $\text{add}(\mathcal{K}_f, \neg \text{isGrasped}(o)),$ $\text{add}(\mathcal{K}_f, \text{isContained}(o)),$ $\text{add}(\mathcal{K}_f, \text{isHandEmpty}(r)),$ $\text{add}(\mathcal{K}_f, \text{containsBearing}(o'))$
assemblePipeTree (robot r, object o, object o')	$K(\text{isRobotGrasps}(r, o)) \wedge$ $K(\neg \text{isGrasped}(o')) \wedge$ $K(\text{isAPipe}(o)) \wedge$ $K(\text{isATree}(o')) \wedge$ $K(\text{containsBearing}(o)) \wedge$ $K(\text{containsBearing}(o')) \wedge$ $K(\neg \text{isContained}(o)) \wedge$ $K(\neg \text{isContained}(o'))$	$\text{del}(\mathcal{K}_f, \text{isRobotGrasps}(r, o)),$ $\text{add}(\mathcal{K}_f, \neg \text{isGrasped}(o)),$ $\text{add}(\mathcal{K}_f, \text{isContained}(o')),$ $\text{add}(\mathcal{K}_f, \text{isHandEmpty}(r)),$ $\text{add}(\mathcal{K}_f, \text{isAGearbox}(o))$

Table 6.7: Evaluation of the BIMANUAL ASSEMBLY scenario. Computation time and procedure calls are measured with respect to the search strategy of the symbolic planner and the type of geometry and collision checking library. Averages are taken over 12 trials, standard deviation is shown in gray.

	Depth-first Search	Breadth-first Search	Iterative Deep- ening Search		
<i>Bounded geometric predicates $\varepsilon < 0.02$ m on a bounding convex decomposition of the scene geometry, using <i>Bullet Physics Library</i></i>					
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		
<i>Variant: Bounded geometric predicates $\varepsilon < 0.02$ m, using <i>Solid</i></i>					
Total Time [s]	0.309 ±0.060	11.721 ±1.826	12.663 ±3.523		
Collision Checking [s]	0.295 ±0.058	11.192 ±1.748	12.060 ±3.317		
<i>Variant: Exact geometric predicates on the original scene geometry, using <i>Bullet Physics Library</i></i>					
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		
<i>Variant: Exact geometric predicates, using the <i>Open Dynamics Engine</i></i>					
Total Time [s]	7.701 ±6.599	480.446 ±62.833	405.665 ±65.473		
Collision Checking [s]	7.681 ±6.590	479.736 ±62.733	405.098 ±65.388		

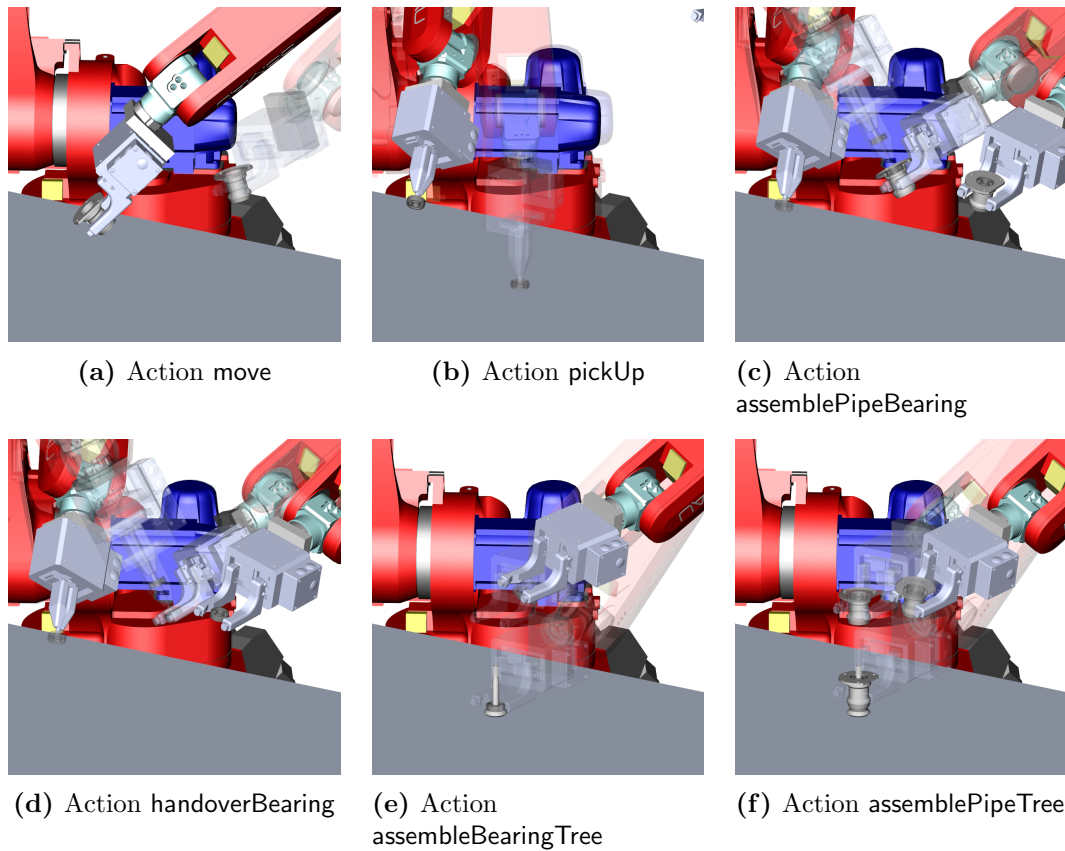


Figure 6.8: Examples of the actions in the BIMANUAL ASSEMBLY scenario. Action `putDown` is not shown; it is the inverse of `pickUp`. In each image, waypoints before the resulting state are shown in transparent colors.

geometric states connected by `move`, `pickUp`, and `putDown` actions until finding the shortest path in depth six. In contrast, depth-first search finds fewer circles in the graph of geometric states. Since it never returns to states before a part has been assembled, it evaluates considerably fewer actions until finding a solution.

6.2.4 Conclusion

In this chapter, we have described the implementation of the Knowledge-level Action and Bounding Geometry Motion planner (KABouM), and studied its performance on a variety of scenarios, from simple pick-and-place tasks to complex assembly actions. Contrary to other approaches to rearrangement and assembly, or classical multi-layered robot control, KABouM finds solutions in the full state space of discrete actions and continuous-valued geometric states. Apart from discrete uncertainty that can be resolved through sensing actions at run-time, we assume full information

of all objects and manipulators. If this condition is met, KABouM can be applied to a wide variety of tasks, ranging from service to manufacturing.

On the one hand, integrated task and motion planning can *generate complex behavior automatically*. It performs a complete search even for complex scenarios, and allows tasks being specified on a domain definition level. As an example, objects are re-grasped when necessary, and even sequences of grasps and bimanual operations are synthesized. Assembly steps are ordered according to their preconditions and effects, rather than by manually-defined plans. Complex geometric conditions and effects are allowed, including reachability with multiple manipulators, avoiding obstacles, but allowing contact of tools and assemblies, and configurations can be specified by geometric constraints. Solving geometric constraints even allows bimanual manipulation to be generated automatically, rather than at predefined waypoints. On the other hand, integrated task and motion planning can *generate correct robot paths from concise domain definitions*, which would be error-prone or even infeasible to program in classical architectures. KABouM verifies all geometric states, even those that may easily be overlooked by a human programmer. As an example, it can coordinate multi-robot motion of diverse actions while avoiding all possible types of collisions.

Chapter 7

Conclusion

In the following, we summarize the main contributions beyond the state of the art: the introduction of single-sided geometric predicates together with the bounding mesh algorithm and their integration in a knowledge-level planning system, capable of solving complex robot manipulation and assembly problems. Finally, we conclude by pointing out applications and further lines of research.

7.1 Contribution

In this thesis, we have addressed the robot task and motion planning problem, in particular its geometric side. First, we provided a formal problem definition in Chapter 3. We began our discussion with symbolic planning on the knowledge level and solved an illustrative problem that allows separate symbolic and geometric planning. In order to solve general problems where symbolic planning cannot be separated from geometric planning, we then discussed the design of geometric predicates and sampling functions, which serve as a symbolic-geometric mapping. Predicates and sampling functions define the interface between both discrete and continuous state spaces, which must be designed carefully to cover the combined search space. Because of this large search space, the efficiency of geometric predicates has a great impact on performance and the size of the problems that can be handled. In order to increase their efficiency, we used the fact that collision and inclusion have asymmetric tolerances; collisions must always be detected, but non-collisions may be ignored up to a certain precision. In particular, we developed a new set of *bounded geometric predicates*, which operate on a single-sided approximation of the geometry (Chapter 4). Note that the idea of single-sided approximation is novel to the field of robotics; prior works are limited to silhouette clipping and ray intersection checks. To achieve this type of approximation, we derived the new *bounding mesh algorithm*. This algorithm generates an approximate mesh that encloses the original geometry at a precision that can be parameterized. At a precision of 1%, the bounding mesh algorithm reduces vertex counts by a factor of 10–20 for the evaluated robot geometries. In conjunction with a bounding convex decomposition of the scene, geometric predicates operate efficiently in complex scenarios. Besides their application to task planning, we discuss the general properties of bounding meshes and bounding convex decomposition. Among these, we observed that a bounding convex decomposition allows collision detection and distance computation within lower and more predictable worst-case time limits than regular meshes. To complete the symbolic-geometric interface, we proposed an expressive, task-level formulation for sampling with geometric constraints (Chapter 5).

Our formulation accepts general constraints, such as coincidence, parallelism, and distance, between robot tool spaces and objects. We then develop a sample-project routine to cover lower-dimensional constraint spaces, which could not be covered by rejection sampling. In Chapter 6, we finally described the software components of the KABouM system. To complete the integration, we defined symbolic-geometric state mapping through a lexicographic order on geometric states.

For evaluation, we demonstrated the effectiveness of the KABouM system in a wide range of scenarios, including dual-arm setups, combinatorially challenging tasks, and bimanual assembly. Among these scenarios was an assembly task with seven different types of transfer and manipulation actions, two of them bimanual, whose solution involves multi-robot collision checking, sequences of re-grasps, and synthesis of bimanual actions. In further experiments, we analyzed different search schemes and show that bounded geometric predicates are more efficient than regular collision checking. In conclusion, we can solve scenarios that are too complex and too error-prone to be programmed manually. Our search progresses in a hybrid state space, which could not be covered by sequential path planning, and verifies complex preconditions, such as multi-robot collisions, which could easily be overlooked by a human programmer.

To conclude, we introduce the idea of single-sided geometric approximation to the field of robotics, implement suitable algorithms, apply this approach to efficient task and motion planning, contribute to sampling with geometric constraints, and finally integrate a system to solve complex task and motion planning problems.

7.2 Future Work

Robot task and motion planning is clearly part of an interdisciplinary area of research between robot path planning, manipulation, computer geometry, and automated planning. The methods proposed in this thesis can likewise be developed further in different lines of research.

Integrated Task and Motion Planning Systems

The described KABouM system (Chapter 6) uses single-sided geometric predicates and constraint space sampling in conjunction with the Planning with Knowledge and Sensing (PKS) planner, which reasons in a modal language of knowledge. It would be interesting to integrate the geometric functions with a different automated planner for comparison. Even though most other planners would not provide reasoning with discrete uncertainty and sensing actions, conventional planners may solve

rearrangement manipulation with different types of heuristics and scale differently with respect to the number of objects.

Besides the integration with additional symbolic planners, additional scenarios and domains can be implemented and evaluated. The scenarios discussed in the evaluation already cover bimanual manipulation and assembly, where task and motion planning can provide an automatic solution to generate robot behavior that would be impractical to program manually. Future work may include a discussion of more complex scenarios with multiple types of objects and containers, larger workspaces, and multi-action logistics or automation tasks, which most evidently require planning on the task level. Even though additional demonstrations would provide little theoretical insight, they would help champion task and motion planning for industrial applications.

Task-level Constraint Formulation

The task-level formulation for geometric constraints between kinematics and objects is inspired by mating operations in computer-aided design software, where constraints are regarded as an intuitive and consistent way to define assemblies of multiple parts. Compared to other formulations in task-level robot programming, most of which are geared towards control [112, 113], our formulation for geometric relations may be a promising approach to more intuitive robot programming by non-experts. As future work, the geometric constraint formulation could be developed further to a robot programming framework that allows non-expert users to specify and solve manipulation tasks.

Bounding Meshes and Bounding Convex Decomposition

The geometric approximation algorithms presented in Chapter 4 follow greedy, locally iterative strategies. When bounding meshes are generated in a pre-processing routine, efficiency is of little concern; a slightly higher approximation quality may potentially be achieved in a global search. For general mesh simplification, several global techniques are known [75]. Some of them may be adapted to single-sided approximation and open a line of research in global bounding mesh optimization. Analogously, bounding convex decomposition can be developed further by applying heuristics used in conventional convex decomposition. Approximate convex decomposition, which is hard to perform efficiently, usually follows divide-and-conquer searches and may likewise be adapted to bounding convex decomposition. At the moment, this is achieved by applying algorithms one after another, but several approximate convex decomposition routines already incorporate conventional mesh

simplification techniques. Integrating both algorithms can potentially improve the quality of the result.

7.3 Further Applications

Similar to future lines of research, applications likewise arise in different areas, including task and motion planning for service and manufacturing, robot control with real-time collision avoidance, and computer graphics. From a general perspective, task and motion planning allows robots to perform abstract tasks automatically and autonomously. It can solve difficult goals, plan in complex geometries, and raise domain specification to the task level. A typical application domain with multifaceted goals is service robotics, where a robot needs to perform different types of actions, including speech, motion, and manipulation. Complex geometries are very common in manufacturing settings, where space is limited and tolerances are small. Solving tasks automatically makes production more flexible and reduces setup times between product variants. Furthermore, task and motion planning can generate efficient plans to coordinate multiple industrial manipulators and use re-grasping and bimanual manipulation behavior that could hardly be programmed by hand. Apart from service and manufacturing robotics, goal definition on the task level also applies to tele-operation of remote robots and robots in space, where limited communication forbids direct feedback control.

Besides intelligent robotics, the individual geometric algorithms of the KABouM system directly apply to computer geometry and real-time collision avoidance. Bounding mesh approximation is in no way limited to the robotics domain, and more generally applies to problems in computer geometry and computer graphics. Most computer geometry algorithms that operate on meshes can be made more efficient when bounding volumes of these meshes are available [75, 79]. Bounding volume hierarchies have long been used, in some cases implicitly, to speed up search and intersection problems. In terms of the bounding volume hierarchy, bounding meshes can be seen as a new, intermediate level between very coarse bounding spheres or bounding boxes on the one hand and the exact mesh on the other hand. In computer geometry, bounding meshes can speed up the broad phase of searches, intersection, and inclusion queries. In search applications, bounding meshes allow fast approximate queries and therefore earlier pruning of search trees, which may further be processed by exact algorithms. For other applications, bounding meshes directly apply to problems that allow a single-sided approximate solution. As an example, clipping operations can directly be performed on bounding meshes, and distance

computation may allow a single-sided approximation in many settings. Apart from computer geometry algorithms, bounding mesh computation can be used interactively in end-user software. In computer graphics software, the bounding mesh operation may serve as a feature to achieve certain visual effects or to simplify a selection of a scene at the user's control.

In conjunction with convex decomposition, bounding convex decomposition makes general meshes accessible to algorithms that operate on convex shapes, while ensuring a single-sided approximation. In general, applications to bounding convex decomposition may include physics simulation, online collision checking, and collision avoidance. Most prominently, bounding convex decomposition allows efficient distance and penetration depth computations that would otherwise be forbiddingly expensive. In contrast to approximate convex decomposition, measured distances and penetration depths are then lower and upper limits. Furthermore, intersection and distance between convex shapes can be computed within low worst-case time limits [100]. Therefore, we can guarantee real-time performance for online robot control and collision avoidance schemes that operate on a bounding convex decomposition of the robot geometry. As a possible application, robot manufacturers can integrate distance computation and collision avoidance in real-time controllers, which are guaranteed effective for environments up to a certain complexity. Bounding convex decomposition more generally applies to collision checking and collision avoidance of pre-processed geometries within guaranteed real-time limits.

Appendix A

Technical Definitions and Proofs

A.1 Definition of the 3D Quadric Metric

We define a quadric metric Q as a function that maps a 3D coordinate $\mathbf{x} \in \mathbb{R}^3$ to a squared distance measure $d^2 \in \mathbb{R} \geq 0$.

$$Q : \mathbf{x} \mapsto \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \quad (\text{A.1})$$

This function can be represented well by a symmetric matrix \mathbf{Q} , which we also refer to as the *quadric* itself. By our definition, a 4-by-4 matrix \mathbf{Q} is a quadric if it is symmetric and $\begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$ is non-negative for all \mathbf{x} . This notation almost equivalent to that by Garland [80]—in more general mathematics terms, Q may be regarded as a non-negative trivariate (or, ternary) quadratic function.

Our main intention to use quadrics is to approximate distances to geometric shapes. In particular, we would like to combine quadrics representing the approximate distance to compounds of geometric shapes. It is therefore useful to define a relation whether one quadric metric is greater or equal than another. Let \succeq define a greater-or-equal relation between quadrics:

$$Q' \succeq Q \quad \text{if and only if} \quad \forall \mathbf{x} : Q'(\mathbf{x}) \geq Q(\mathbf{x}) \quad (\text{A.2})$$

From these definitions, we can see that all symmetric, positive-semidefinite matrices are quadrics, that the quadric error metric is closed under matrix addition, and that \succeq is a partial order. (\succeq is reflexive, antisymmetric, and transitive, but not total.)

A.1.1 Distances to Geometric Primitives

Quadric metrics can express the exact squared distance to the 3D geometric primitives points, lines, and planes. We would like to note that our definition of *quadric metrics* is different from the more generally used quadric surfaces or quadric surfaces in projective space [129, pp. 73ff.], in which $Q(\mathbf{x})$ may be negative. While the locus of zeros of those quadrics can define surfaces of more general shapes such as cylinders and ellipsoids, they are not defined as non-negative and therefore cannot be used as distance metrics to geometric shapes.

For our definition of quadric metrics, the squared distances to a point \mathbf{p} , a line (\mathbf{p}, \mathbf{l}) with a direction \mathbf{l} and a plane (\mathbf{p}, \mathbf{n}) with a normal \mathbf{n} are given as follows:

$$\mathbf{Q}_{\text{point}}(\mathbf{p}) = \begin{bmatrix} \mathbf{1} & -\mathbf{p} \\ -\mathbf{p}^T & \mathbf{p}^2 \end{bmatrix} \quad (\text{A.3})$$

$$\mathbf{Q}_{\text{line}}(\mathbf{p}, \mathbf{l}) = \mathbf{A}^T \mathbf{A} \quad \text{with} \quad \mathbf{A} = \text{nullspace}(\mathbf{l})^T \begin{bmatrix} \mathbf{1} & -\mathbf{p} \end{bmatrix} \quad (\text{A.4})$$

$$\mathbf{Q}_{\text{plane}}(\mathbf{p}, \mathbf{n}) = \mathbf{A}^T \mathbf{A} \quad \text{with} \quad \mathbf{A} = \mathbf{n}^T \begin{bmatrix} \mathbf{1} & -\mathbf{p} \end{bmatrix} \quad (\text{A.5})$$

From these primitive shapes, quadric metrics for more complex, compound shapes can be generated. The simplest way to define the quadric metric for an arbitrary compound shape $A \cup B$ is of course the addition.

$$Q_{A \cup B} := Q_A + Q_B \succeq \min(Q_A, Q_B) \succeq d^2(A \cup B) \quad (\text{A.6})$$

Assuming that Q_A and Q_B are upper bounds of the squared distances to A and B , we can construct an upper bound $Q_{A \cup B}$ for the squared distance to the compound $A \cup B$. This simple addition is however too coarse an estimate for distance queries to geometric meshes (see Figure 4.4d); more refined alternatives are discussed in Section 4.3.3.

A.1.2 Operations on Quadric Metrics

Before we derive a locally tight approximation to compound shapes distance by quadrics, we first need to define a number of basic operations on quadric metrics used in further discussion. For an in-depth discussion, more geometric properties of quadric metrics are covered in a thesis by Garland [72, pp. 61ff.].

Transformation A quadric can be rotated and translated in 3D space. To apply a transformation $\mathbf{T} \in \text{SE}(3)$ to a quadric \mathbf{Q} , we can compute $\mathbf{T}^T \mathbf{Q} \mathbf{T}$.

Minimum Since the function Q is continuous and non-negative, it has a minimum. The minimizer $\operatorname{argmin}(Q)$ is a linear subspace depending on the quadratic terms; it is most often a minimizing point, but may be more generally a line, a plane, or the whole 3D space. Since a minimizer \mathbf{m} fulfills a derivative of zero in the quadratic function of the quadric, it can be solved from the linear equation

$$\mathbf{Q}_{1:3,1:3} \mathbf{m} + \mathbf{Q}_{1:3,4} = \mathbf{0} \quad (\text{A.7})$$

It should be noted that a quadric is often singular—an example would be the distance to a plane Q_{plane} , which has rank 1—therefore, solving Eq. A.7 requires numerically stable algorithms such as singular value decomposition.

Factorization In our definition of quadrics, there exists a transformation \mathbf{T} such that a quadric \mathbf{Q} becomes a non-negative diagonal matrix, which we name \mathbf{S} . In other words, \mathbf{Q} can be factorized into $\mathbf{T}^T \mathbf{S} \mathbf{T}$. Geometrically, we can transform the quadric metric such that its minimum is at the origin, and its value increases quadratically along the unit axes. Surfaces of equal values (isosurfaces) of Q then become, in general, axis-aligned ellipsoids. To perform this factorization, we first extract a translation $\mathbf{T}_{\text{trans}}$ to move the minimum to the origin. This translation is given by

$$\mathbf{T}_{\text{trans}} = \begin{bmatrix} \mathbf{1} & -\operatorname{argmin}(Q) \\ \mathbf{0}^T & 1 \end{bmatrix}, \quad (\text{A.8})$$

and leads to the intermediate factorization $\mathbf{Q} = \mathbf{T}_{\text{trans}}^T \mathbf{A} \mathbf{T}_{\text{trans}}$ to a quadric \mathbf{A} that has no linear term, i.e. $\mathbf{A}_{1:3,4}^T = \mathbf{A}_{4,1:3} = \mathbf{0}$. After that, we can perform a singular value decomposition on the quadratic part of the origin-aligned quadric, factorizing $\mathbf{A}_{1:3,1:3} = \mathbf{V}^T \mathbf{S}_{1:3,1:3} \mathbf{V}$. The transformation \mathbf{T} for the whole factorization is then given by

$$\mathbf{T} = \begin{bmatrix} \mathbf{V} & -\mathbf{V} \operatorname{argmin}(Q) \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (\text{A.9})$$

From the normalized quadric \mathbf{S} , we deduce some of its properties. The first three diagonal entries of \mathbf{S} are the inverted axis lengths of the unit value ellipsoid. If one or more entries are zero, areas of equal values, or isosurfaces, degenerate to an infinite elliptic cylinder or two infinite planes. The fourth entry $S_{4,4}$ is the constant cost term of the quadric.

A.2 Convexity Invariance of the Bounding Mesh Algorithm

For a simplified cost function $E_{\text{triangles}}$ that is a weighted sum of squared distances to neighboring vertices (Eq. A.3), we can observe that the bounding mesh algorithm preserves the convex property of shapes in non-degenerate cases. Given a convex input mesh M , the bounding mesh algorithm (Section 4.2, Algorithm 2) will then generate a convex bounding mesh M' .

Sketch of Proof. It is sufficient to show the invariance of the convex property only for a single edge contraction, because modifications to the mesh are only made within the edge contraction step and the invariance is then preserved by induction. We assume that the cost function $E(e, \mathbf{v})$ of all edges e is a weighted sum of squared distances from \mathbf{v} to points within the convex hull of the neighborhood $P(e)$, $\text{hull}(P(e))$. We may also assume that $\text{hull}(P(e))$ does not intersect triangles outside of $P(e)$. Because of the bounding mesh constraints, any constraint minimizer v of $E(e)$ is clearly outside of M . However, an unconstrained minimizer of $E(e)$ is a point v^* within $\text{hull}(P(e))$. Therefore, at least one constraint must hold with equality for the constrained minimizer, let us name this plane constraint p . Also, let $\bar{P}(e)$ denote all triangle planes outside the neighborhood $P(e)$. Now suppose the optimal contraction point v would be outside of the simplex defined by $\bar{P}(e)$. Note that all intersection points of planes from $P(e)$ are located within this simplex. v is therefore not fully constrained and may move along a constraint subspace. From the convexity of $\text{hull}(P(e))$, it could therefore be moved closer to the unconstrained minimizer v^* and have lower costs. This contradicts our assumption that an optimal contraction point outside the simplex of $\bar{P}(e)$ could be generated. As a result, the contracted mesh is convex. Note that cost functions are updated in a way that unconstrained minima stay within $\text{hull}(P(e))$, so assumptions also hold for all subsequent edge contractions. \square

Note that the above argument only applies to a cost function that is a sum of squared distances to points within an edge neighborhood. If we choose a cost function that includes other terms, for instance squared distances to planes, convexity is, in general, no longer preserved. It is then straightforward to construct a counterexample where the projection onto \bar{p} would increase costs and a concave edge is introduced, or to find concave edges in the real output of the software implementation.

A.3 Closed-form Inverse Kinematics

To provide more efficient constraint space sampling strategies in a number of the scenarios discussed, we derived closed-form solutions for a range of manipulators with five and six rotational axes. We use a variant of Denavit-Hartenberg parameters defined by Khalil and Kleinfinger [130]. Note that an implementation of the following closed-form solutions should check for cases where solution is not defined and the pose is not reachable, such as $\arccos(x)$ with $\|x\| > 1$.

Inverse Position for Five Revolute Axes

Given is a kinematic chain of five rotational axes with arbitrary Denavit-Hartenberg parameters d_1, a_2, a_3, d_5 , fixed parameters $\alpha_1 = -\pi/2, \alpha_3 = \pi/2$, and all other parameters set to zero. Since this kinematic chain does not cover SE(3) in general, it is favorable to solve the inverse position $\boldsymbol{\theta}$ for given tool center position \mathbf{x} , pitch β , and roll ϕ . A solution may then be obtained as follows.

$$\begin{aligned}\theta_0 &= \text{atan2}(x_1, x_0) \\ \mathbf{t}_3 &= \begin{bmatrix} 0 & 0 & 1 \\ \cos(\theta_0) & \sin(\theta_0) & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 - d_1 \end{bmatrix} - d_5 \begin{bmatrix} \cos(\pi/2 - \beta) \\ \sin(\pi/2 - \beta) \end{bmatrix} \\ \theta_1 &= \text{atan2}(t_{3,1}, t_{3,0}) + \arccos((a_2^2 + \mathbf{t}_3^2 - a_3^2) / (2a_2 \|\mathbf{t}_3\|)) \\ \theta_2 &= \pi - \arccos((a_2^2 + a_3^2 - \mathbf{x}_3^2) / (2a_2 a_3)) \\ \theta_3 &= \pi - \beta - \theta_1 - \theta_2 \\ \theta_4 &= \phi\end{aligned}$$

Inverse Position for Six Revolute Axes

Given is a kinematic chain of six rotational axes with arbitrary Denavit-Hartenberg parameters $a_1, a_2, a_3, d_1, d_3, d_4, d_6$, fixed parameters $\alpha_0 = -\pi/2, \alpha_2 = -\pi/2, \alpha_3 = \pi/2, \alpha_4 = \pi/2$, and all other parameters set to zero; $\boldsymbol{\theta}$ is to be found for a given end-effector pose (\mathbf{R}, \mathbf{x}) . The following solution is a generalization of [131, p. 72ff.] to allow $a_1 \neq 0$. Up to eight individual solutions may be obtained by choosing different signs $k_0, k_1, k_2 \in \{1, -1\}$.

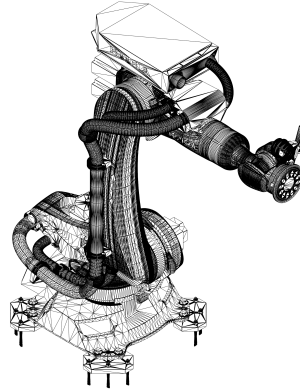
$$\begin{aligned}\mathbf{t}_4 &= \mathbf{x} - d_6 \mathbf{R}_{0:2,2} \\ r &= \sqrt{t_{4,0}^2 + t_{4,1}^2 - d_5^2}\end{aligned}$$

$$\begin{aligned}
\theta_0 &= \text{atan2}(-k_0 r t_{4,1} - d_3 t_{4,0}, -k_0 r t_{4,0} + d_3 t_{4,1}) \\
s_0 &= \cos(\theta_0) t_{4,0} + \sin(\theta_0) t_{4,1} - a_1 \\
s_1 &= t_{4,2} - d_1 \\
u &= \sqrt{a_3^2 + d_4^2} \\
v &= \sqrt{s_0^2 + s_1^2} \\
\theta_1 &= k_0 k_1 \arccos((a_2^2 + v^2 - u^2)/(2a_2 v)) - \text{atan2}(s_1, s_0) \\
\theta_2 &= k_0 k_1 \arccos((a_2^2 + u^2 - v^2)/(2a_2 u)) - \text{atan2}(d_4, a_3) - \pi \\
e &= \cos(\theta_1 + \theta_2 + \pi) \\
f &= \sin(\theta_1 + \theta_2 + \pi) \\
\theta_3 &= \text{atan2}\left(k_2 \begin{bmatrix} -\sin(\theta_0) & \cos(\theta_0) & 0 \end{bmatrix} \mathbf{R}_{0:2,2}, k_2 \begin{bmatrix} \cos(\theta_0)e & \sin(\theta_0)e & -f \end{bmatrix} \mathbf{R}_{0:2,2}\right) \\
\theta_4 &= \text{atan2}\left(\begin{bmatrix} e \cos(\theta_0) \cos(\theta_3) - \sin(\theta_0) \sin(\theta_3) \\ e \sin(\theta_0) \cos(\theta_3) + \cos(\theta_0) \sin(\theta_3) \\ -f \cos(\theta_3) \end{bmatrix}^{\text{T}} \mathbf{R}_{0:2,2}, \begin{bmatrix} f \cos(\theta_0) \\ f \sin(\theta_0) \\ e \end{bmatrix}^{\text{T}} \mathbf{R}_{0:2,2}\right) + \pi \\
\theta_5 &= \text{atan2}\left(\begin{bmatrix} -e \cos(\theta_0) \sin(\theta_3) - \sin(\theta_0) \cos(\theta_3) \\ -e \sin(\theta_0) \sin(\theta_3) + \cos(\theta_0) \cos(\theta_3) \\ f \sin(\theta_3) \end{bmatrix}^{\text{T}} \mathbf{R}_{0:2,0}, \begin{bmatrix} -e \cos(\theta_0) \sin(\theta_3) - \sin(\theta_0) \cos(\theta_3) \\ -e \sin(\theta_0) \sin(\theta_3) + \cos(\theta_0) \cos(\theta_3) \\ f \sin(\theta_3) \end{bmatrix}^{\text{T}} \mathbf{R}_{0:2,1}\right)
\end{aligned}$$

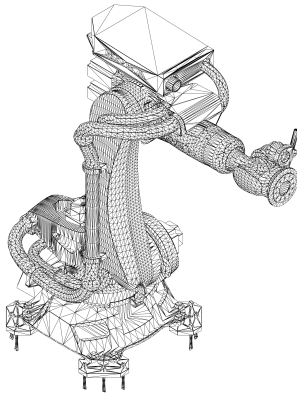
Appendix B

Bounding Mesh Evaluation

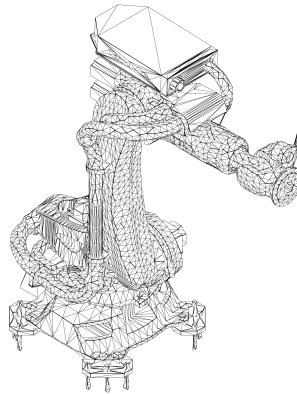
B.1 Additional Bounding Mesh Examples



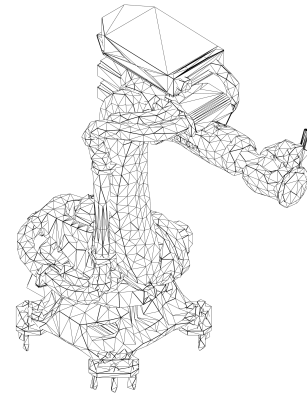
(a) Original mesh with 251,117 vertices



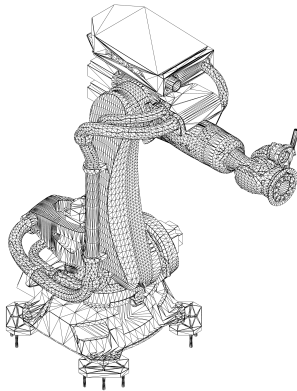
(b) Bounding mesh with 20,000 vertices



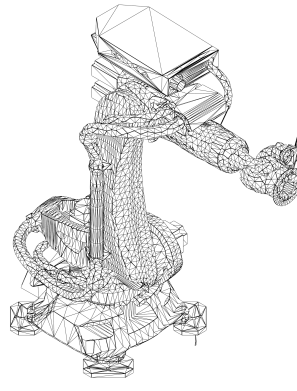
(c) Bounding mesh with 10,000 vertices



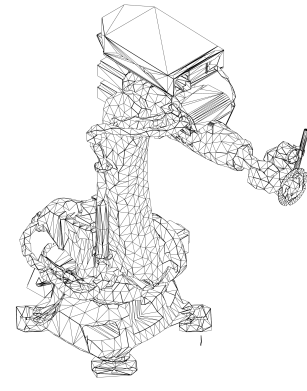
(d) Bounding mesh with 5,000 vertices



(e) Inner bounding mesh with 20,000 vertices

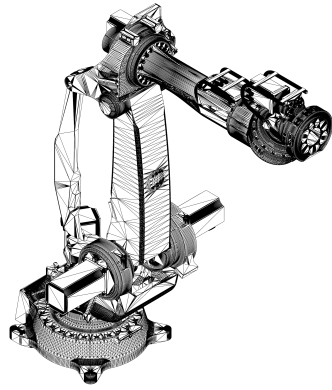


(f) Inner bounding mesh with 10,000 vertices

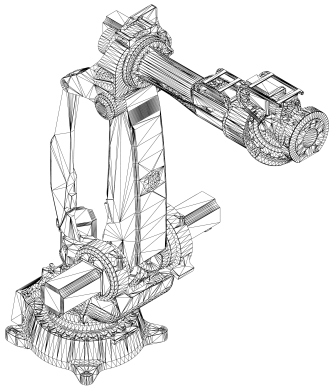


(g) Inner bounding mesh with 5,000 vertices

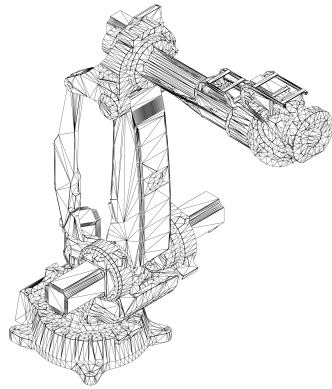
Figure B.1: Several outer and inner bounding meshes of a KUKA robot



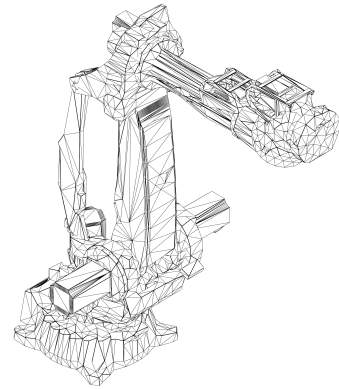
(a) Original mesh with 241,190 vertices



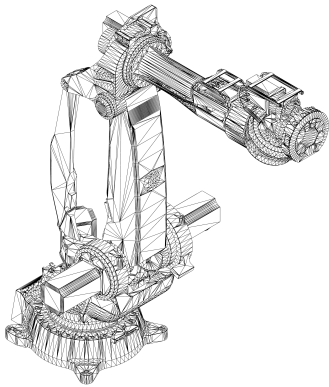
(b) Bounding mesh with 20,000 vertices



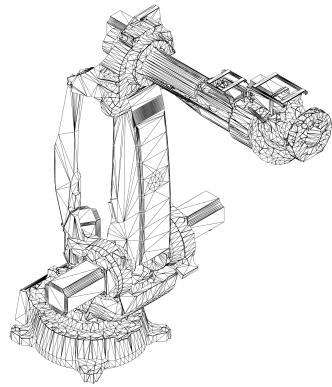
(c) Bounding mesh with 10,000 vertices



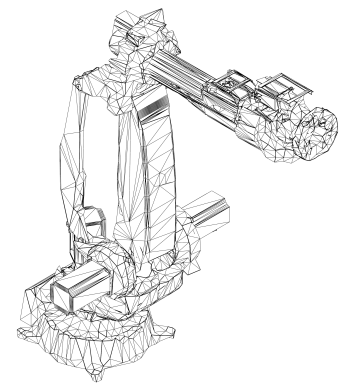
(d) Bounding mesh with 5,000 vertices



(e) Inner bounding mesh with 20,000 vertices



(f) Inner bounding mesh with 10,000 vertices



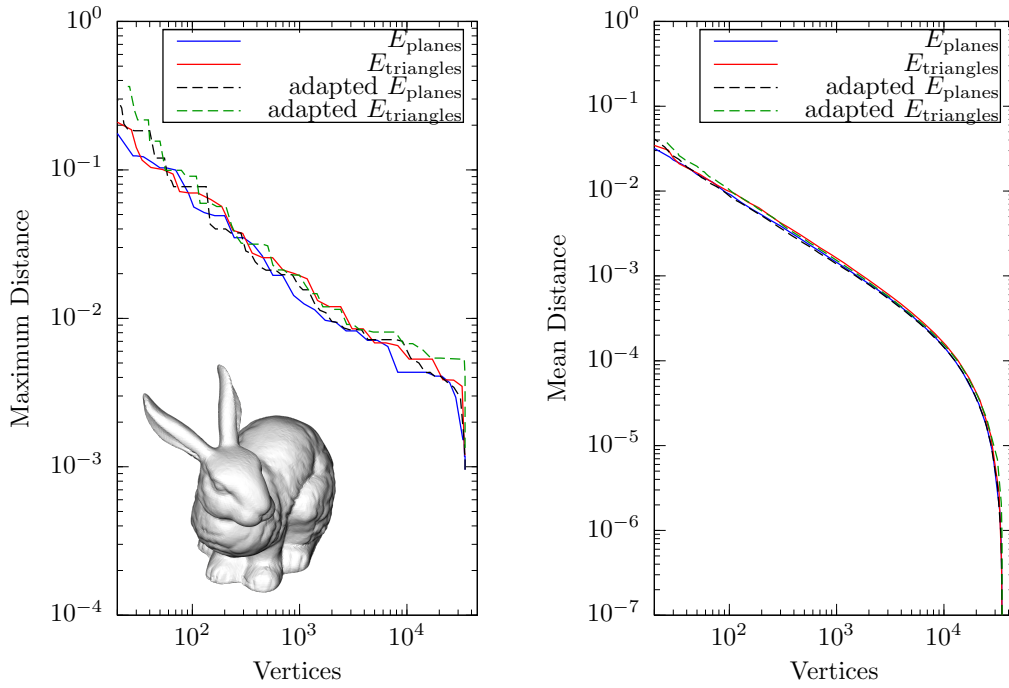
(g) Inner bounding mesh with 5,000 vertices

Figure B.2: Several outer and inner bounding meshes of a COMAU robot.

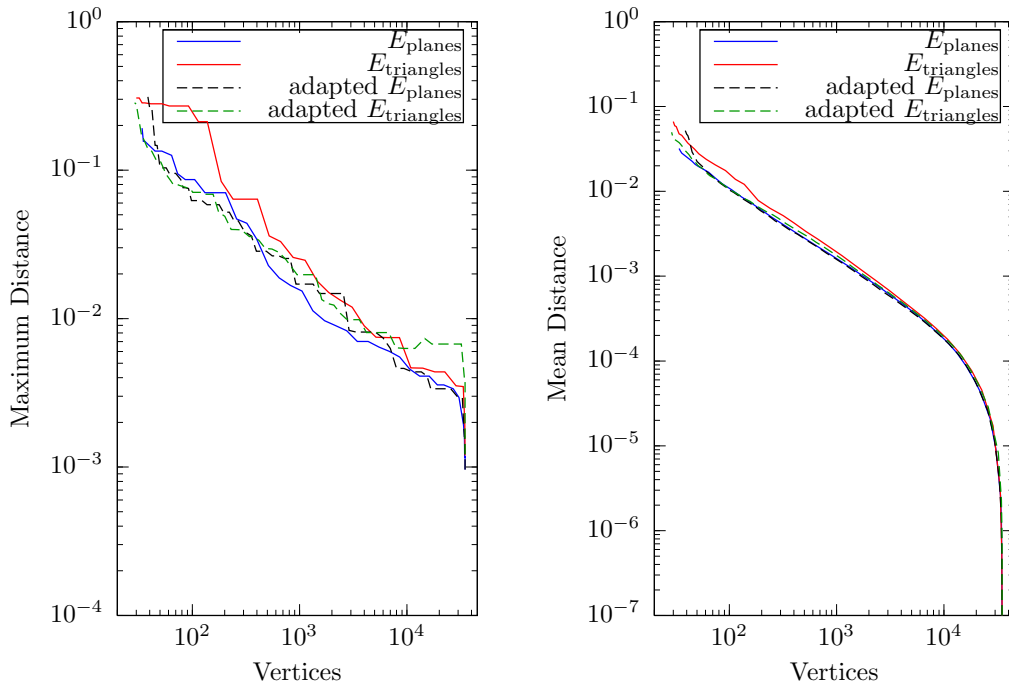
B.2 Evaluation of Cost Functions for Bounding Mesh Decimation

Table B.1: Comparison of cost functions for bounding mesh decimation.

Cost function	$E_{\text{planes, plane distances}}$				Adapted $E_{\text{planes, plane distances}}$ with adaptation of the constant term				$E_{\text{triangles, triangle center distances}}$				
	Distance from original to approximation [mm]	max	mean	max	Distance from original to approximation [mm]	max	mean	max	Distance from original to approximation [mm]	max	mean	max	
Number of vertices n													
MITSUBISHI, original triangulation, bounding box diagonal: 1239.37 mm 108 419 (Figure 4.7a)													
20 000	0.016	0.115	0.005	0.140	0.082	1.795	0.262	3.413	0.033	0.674	0.061	1.899	
10 000	0.053	1.371	0.021	0.458	0.222	3.018	0.580	6.215	0.097	1.208	0.154	2.999	
5 000	0.139	1.371	0.071	1.830	0.546	5.858	1.239	9.048	0.220	2.981	0.329	6.729	
2 500	0.370	3.473	0.202	4.807	1.284	12.522	2.595	20.084	0.523	4.101	0.632	7.887	
KUKA, original triangulation, bounding box diagonal: 3443.75 mm 251 117 (Figure B.1)													
20 000	0.448	26.217	0.247	18.150	0.636	7.685	0.934	22.871	0.701	6.834	0.698	22.513	
10 000	1.274	26.217	0.749	19.420	2.066	18.070	2.733	24.920	1.697	12.628	1.631	25.258	
5 000	3.428	46.441	2.167	44.752	4.900	30.004	5.890	52.727	3.750	21.541	3.757	35.524	
2 500	9.851	310.275	5.789	85.162	10.793	75.640	12.527	81.235	9.077	53.103	8.529	61.689	
COMAU, original triangulation, bounding box diagonal: 3719.08 mm 241 190 (Figure B.2)													
20 000	0.155	2.058	0.115	3.757	0.335	6.704	0.984	18.010	0.158	3.299	0.363	12.575	
10 000	0.723	6.065	0.637	16.010	1.320	15.021	3.223	43.363	0.718	9.153	1.318	40.288	
5 000	2.096	23.480	2.180	43.025	3.672	23.489	6.547	49.266	2.060	16.673	3.115	44.771	
2 500	6.143	78.253	5.427	50.974	7.909	42.383	11.053	70.658	5.384	39.394	6.466	59.966	

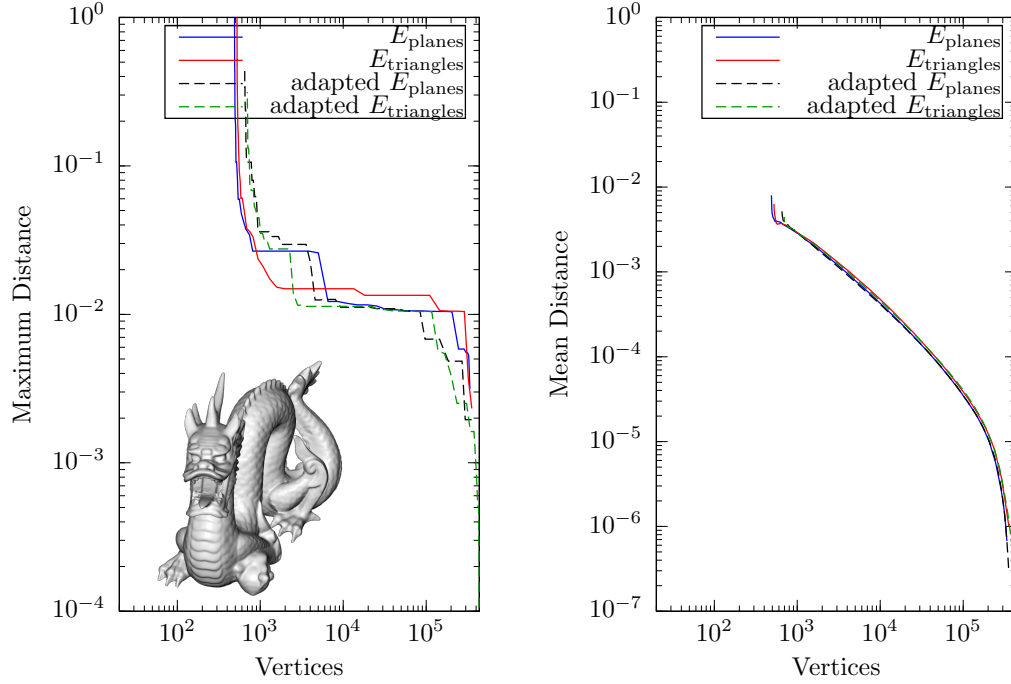


(a) Outer bounding meshes

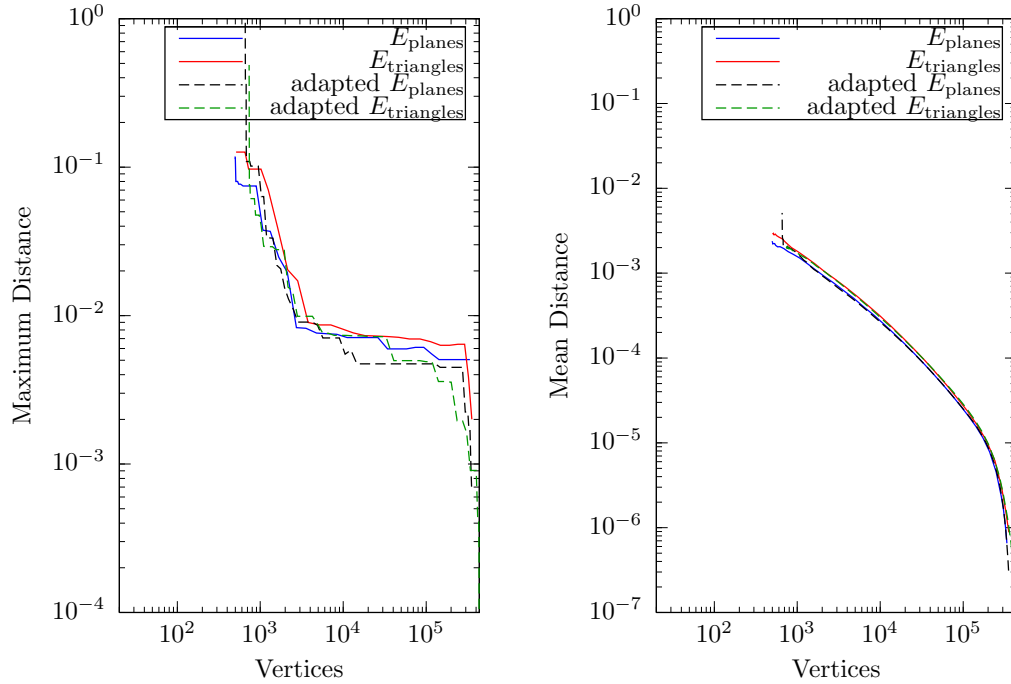


(b) Inner bounding meshes

Figure B.3: Hausdorff distance of a series of bounding meshes of the STANFORD BUNNY with respect to cost functions. The STANFORD BUNNY is a 3D scanned model of 34,835 vertices, with a number of holes in its topology. Distances are given in relation to the bounding box diagonal. Note the logarithmic axes.

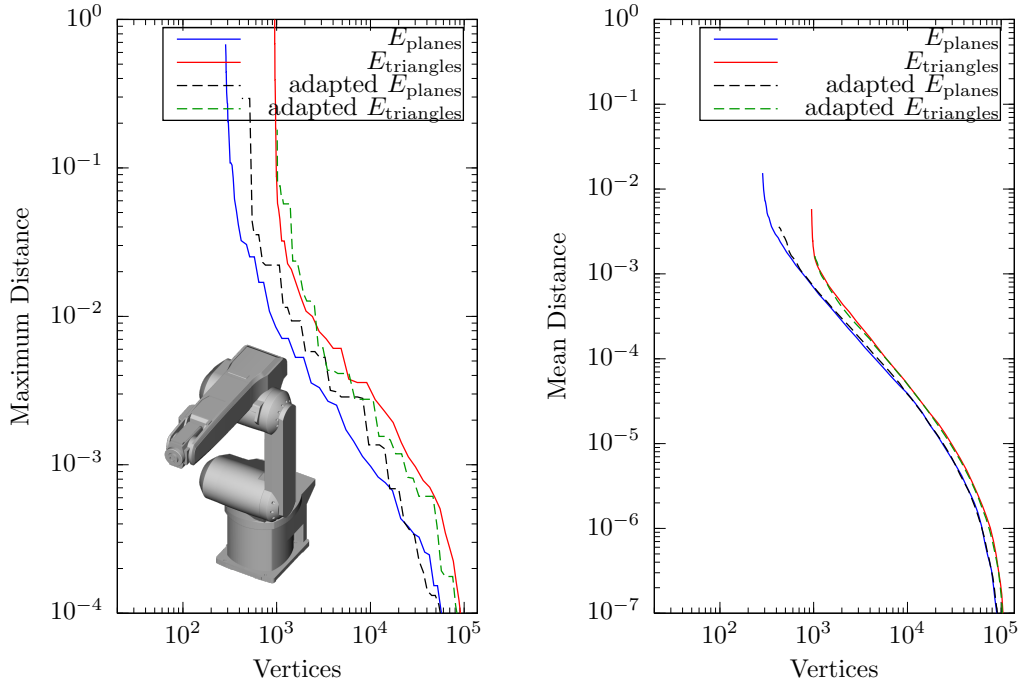


(a) Outer bounding meshes

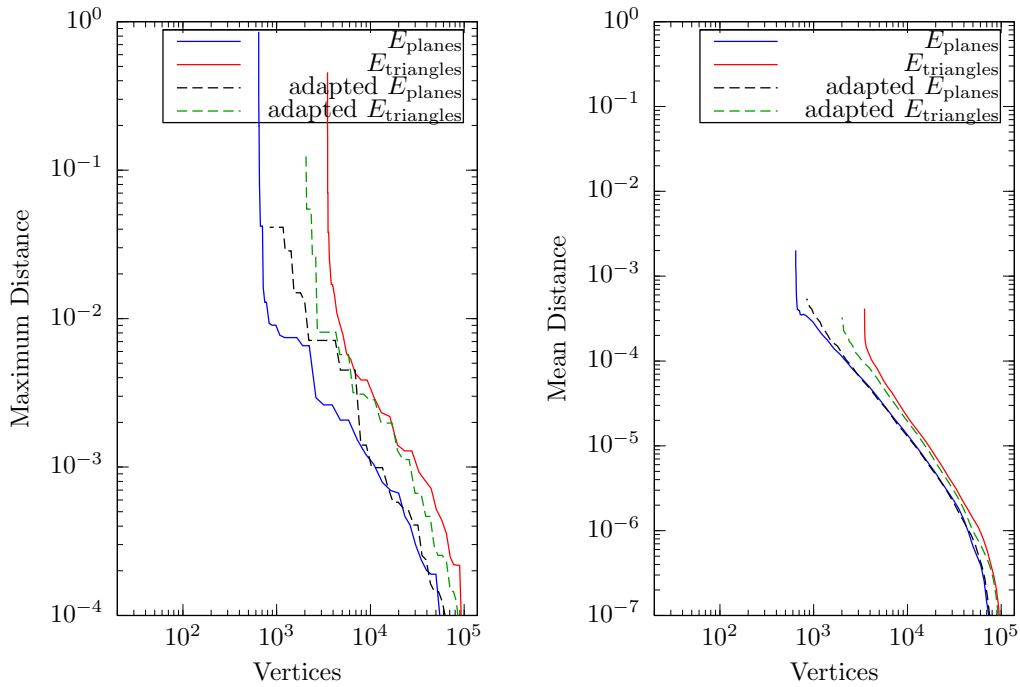


(b) Inner bounding meshes

Figure B.4: Hausdorff distance of a series of bounding meshes of the DRAGON model with respect to cost functions. The DRAGON is a 3D scanned and reconstructed model of 437,645 vertices. Distances are given in relation to the bounding box diagonal. Note the logarithmic axes.

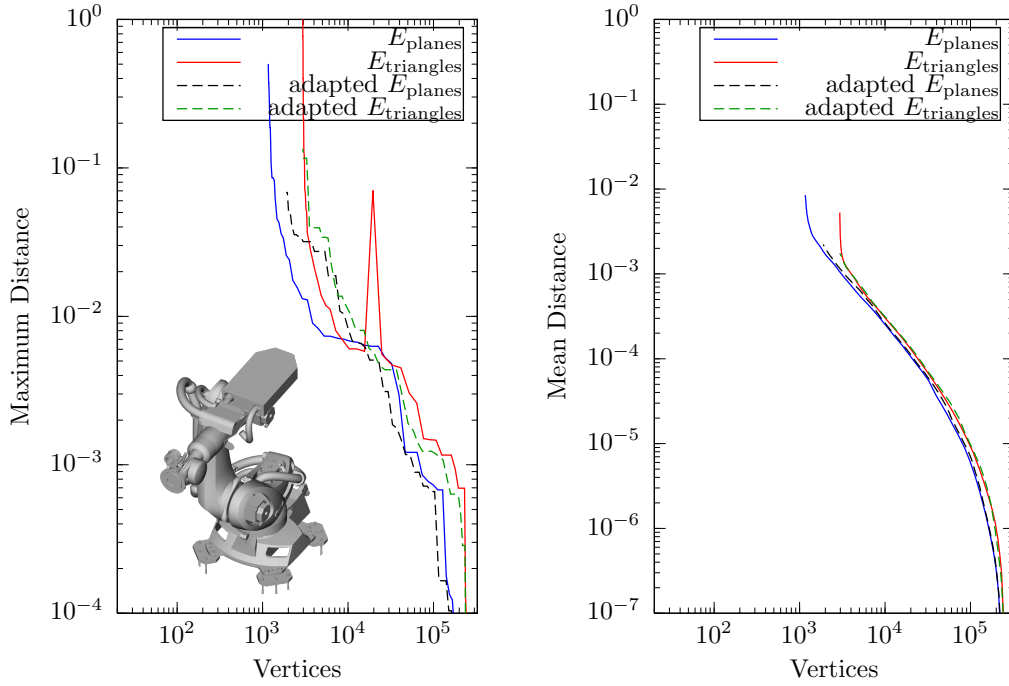


(a) Outer bounding meshes

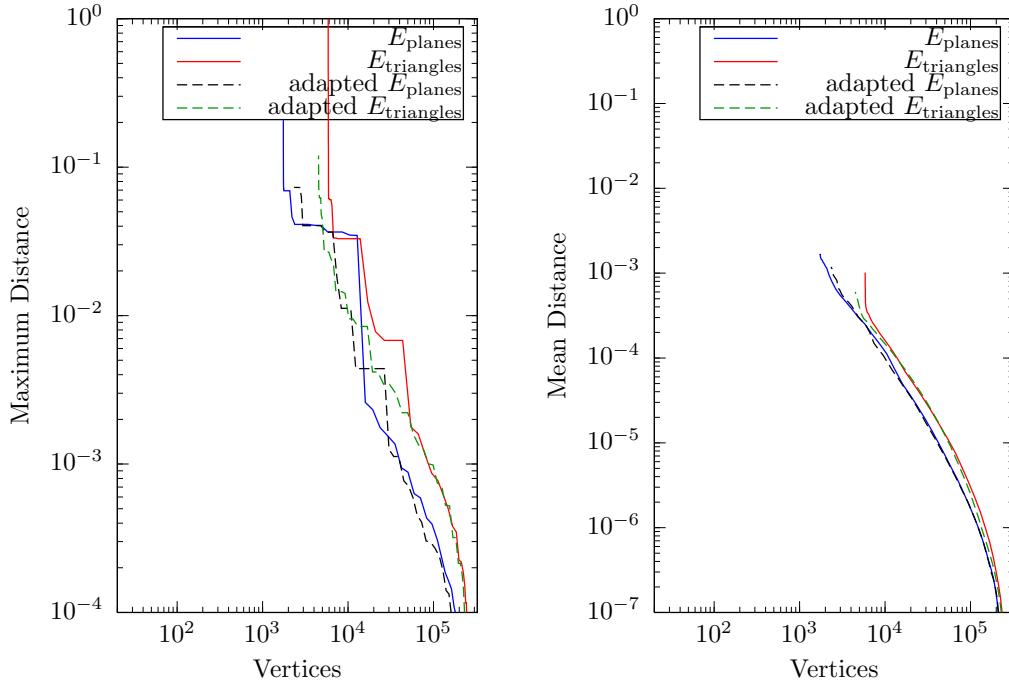


(b) Inner bounding meshes

Figure B.5: Hausdorff distance of a series of bounding meshes of the MITSUBISHI model with respect to cost functions. The MITSUBISHI model is a triangulated mesh of 108,419 vertices. Distances are given in relation to the bounding box diagonal. Note the logarithmic axes.

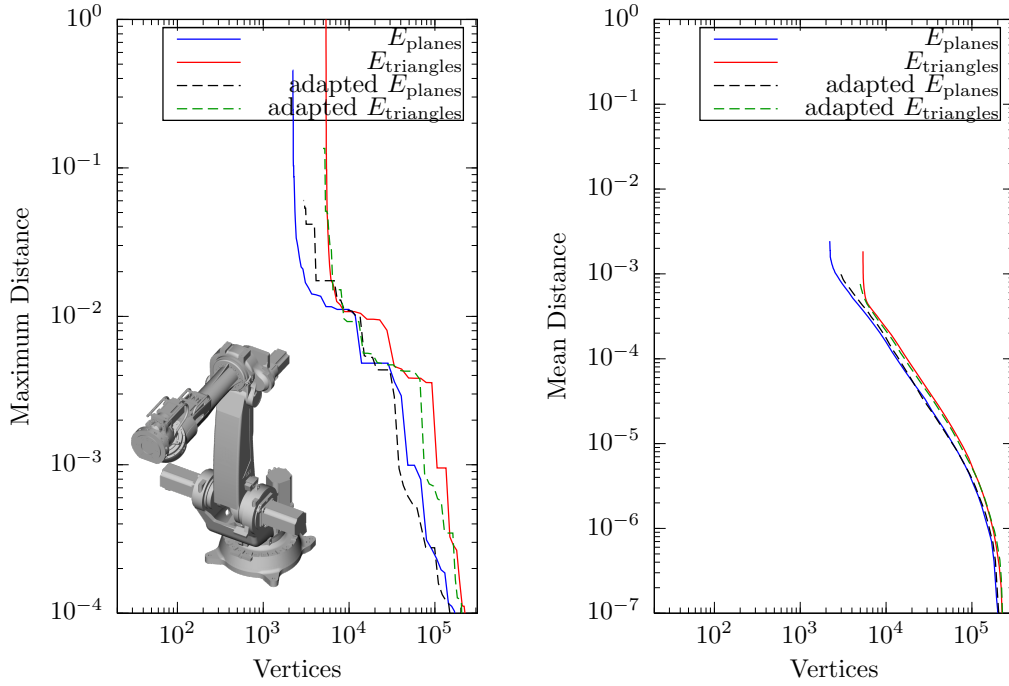


(a) Outer bounding meshes

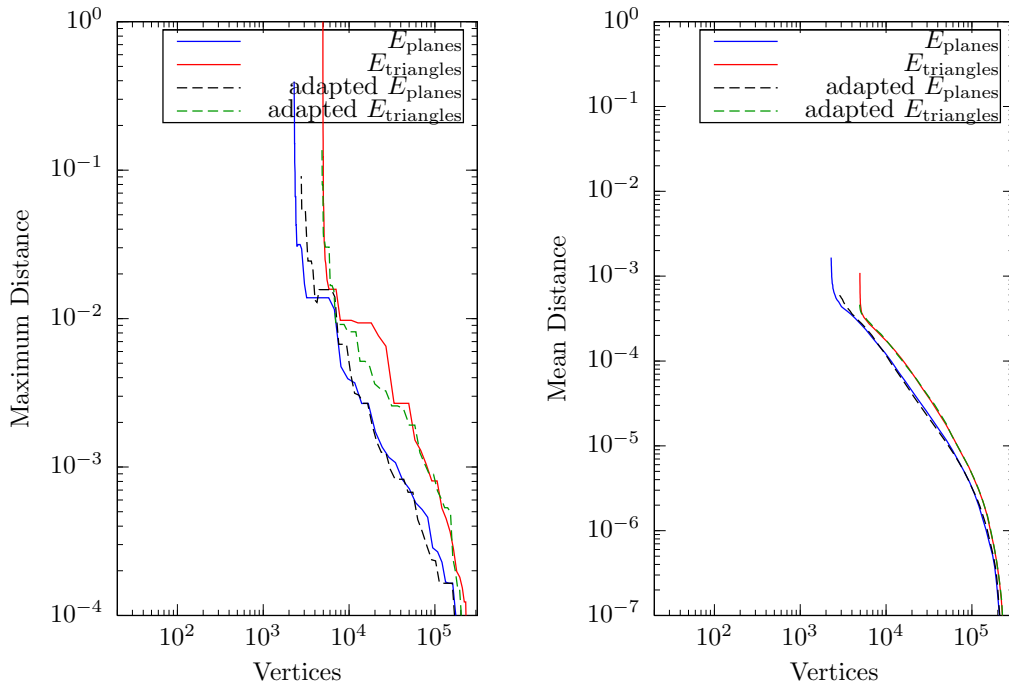


(b) Inner bounding meshes

Figure B.6: Hausdorff distance of a series of bounding meshes of the KUKA model with respect to cost functions. The KUKA model is a triangulated mesh of 251,117 vertices with many round details. Distances are given in relation to the bounding box diagonal. Note the logarithmic axes.



(a) Outer bounding meshes



(b) Inner bounding meshes

Figure B.7: Hausdorff distance of a series of bounding meshes of the COMAU model with respect to cost functions. The COMAU model is a triangulated mesh of 241,190 vertices with many details. Distances are given in relation to the bounding box diagonal. Note the logarithmic axes.

Bibliography

- [1] An Oxford Graduate, *Observations on the Automaton Chess Player*, J. Hatchard, 1819.
- [2] C. Matuszek, B. Mayton, R. Aimi, M. P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J. R. Smith, D. Fox, *Gambit: An autonomous chess-playing robotic system*, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 4291–4297.
- [3] F. Gravot, S. Cambon, R. Alami, *aSyMov: A Planner That Deals with Intricate Symbolic and Geometric Problems*, in: *International Symposium on Robotics Research (ISRR)*, 2003, pp. 100–110.
- [4] S. Cambon, R. Alami, F. Gravot, *A hybrid approach to intricate motion, manipulation and task planning*, *International Journal of Robotics Research (IJRR)* 28 (1) (2009) 104–126.
- [5] C. Dornhege, *Task Planning for High-Level Robot Control*, Dissertation, Albert-Ludwigs-Universität Freiburg (2014).
- [6] A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, A. Knoll, *KVP: A Knowledge of Volumes Approach to Robot Task Planning*, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 202–208.
- [7] A. Gaschler, R. P. A. Petrick, T. Kröger, A. Knoll, O. Khatib, *Robot Task Planning with Contingencies for Run-time Sensing*, in: *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Combining Task and Motion Planning*, 2013.
- [8] A. Gaschler, R. P. A. Petrick, T. Kröger, O. Khatib, A. Knoll, *Robot Task and Motion Planning with Sets of Convex Polyhedra*, in: *Robotics: Science and Systems (RSS) Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, 2013.

- [9] A. Gaschler, S. Nogina, R. P. A. Petrick, A. Knoll, Planning perception and action for cognitive mobile manipulators, in: SPIE Volume 9025 – Intelligent Robots and Computer Vision XXXI: Algorithms and Techniques, 2014.
- [10] R. P. A. Petrick, A. Gaschler, Extending Knowledge-Level Contingent Planning to Robot Task Planning, in: International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (Plan-Rob), 2014.
- [11] A. Gaschler, I. Kessler, R. P. A. Petrick, A. Knoll, Extending the Knowledge of Volumes Approach to Robot Task Planning with Efficient Geometric Predicates, in: IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 3061–3066.
- [12] A. Gaschler, Q. Fischer, A. Knoll, The Bounding Mesh Algorithm, Tech. Rep. TUM-I1522, Technische Universität München, Germany (June 2015).
- [13] R. P. A. Petrick, A. Gaschler, Knowledge-level planning for robot task planning and human-robot interaction, in: RSS Workshop on Combining AI Reasoning and Cognitive Science with Robotics, 2015.
- [14] N. Somani, A. Gaschler, M. Rickert, A. Perzylo, A. Knoll, Constraint-based task programming with CAD semantics: From intuitive specification to real-time control, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015.
- [15] S. Russell, P. Norvig, Artificial Intelligence, A modern approach, 3rd Edition, Prentice-Hall, Pearson Education, 2010.
- [16] N. J. Nilsson, A mobile automaton: An application of artificial intelligence techniques, Tech. rep. (1969).
- [17] R. E. Fikes, N. J. Nilsson, STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence* 2 (1971) 189–208.
- [18] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O’Donnell, Task-level planning of pick-and-place robot motions, *Computer* 22 (3) (1989) 21–29.
- [19] T. Lozano-Pérez, J. L. Jones, P. A. O’Donnell, E. Mazer, *Handey: a robot task planner*, MIT Press, 1992.
- [20] L. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Transactions on Robotics and Automation* 12 (4) (1996) 566–580.

- [21] S. M. LaValle, J. J. Kuffner, Randomized Kinodynamic Planning, *International Journal of Robotics Research (IJRR)* 20 (5) (2001) 378–400.
- [22] C. Dornhege, M. Gissler, M. Teschner, B. Nebel, Integrating symbolic and geometric planning for mobile manipulation, in: *IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR)*, 2009, pp. 1–6.
- [23] L. P. Kaelbling, T. Lozano-Pérez, Unifying Perception, Estimation and Action for Mobile Manipulation via Belief Space Planning, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 2952–2959.
- [24] M. Rickert, Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems, *Dissertation, Technische Universität München* (2011).
- [25] E. Plaku, G. D. Hager, Sampling-based motion planning with symbolic, geometric, and differential constraints, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 5002–5008.
- [26] S. Srivastava, E. Fang, R. Lorenzo, R. Chitnis, S. Russell, P. Abbeel, Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 639–646.
- [27] L. P. Kaelbling, T. Lozano-Pérez, Integrated Task and Motion Planning in Belief Space, *International Journal of Robotics Research (IJRR)* 32 (9–10) (2013) 1194–1227.
- [28] J. Barry, L. P. Kaelbling, T. Lozano-Pérez, A Hierarchical Approach to Manipulation with Diverse Actions, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 1799–1806.
- [29] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, B. Nebel, Semantic Attachments for Domain-Independent Planning Systems, in: *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009, pp. 114–121.
- [30] N. J. Nilsson, *Shakey The Robot*, Tech. Rep. 323, AI Center, SRI International (Apr. 1984).
- [31] L. P. Kaelbling, T. Lozano-Pérez, Integrated robot task and motion planning in the now, Tech. Rep. MIT-CSAIL-TR-2012-018, MIT (2012).

- [32] R. Dearden, C. Burbridge, An Approach for Efficient Planning of Robotic Manipulation Tasks, in: International Conference on Automated Planning and Scheduling (ICAPS), 2013.
- [33] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, T. Uras, Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation, in: IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 4575–4581.
- [34] K. Hauser, V. Ng-Thow-Hing, Randomized multi-modal motion planning for a humanoid robot manipulation task, International Journal of Robotics Research (IJRR) 30 (6) (2011) 678–698.
- [35] D. Leidner, A. Dietrich, F. Schmidt, C. Borst, A. Albu-Schäffer, Object-Centered Hybrid Reasoning for Whole-Body Mobile Manipulation, in: IEEE International Conference on Robotics and Automation (ICRA), 2014.
- [36] J. Bidot, L. Karlsson, F. Lagriffoul, A. Saffiotti, Geometric Backtracking for Combined Task and Path Planning in Robotic Systems, Artificial Intelligence In press.
- [37] J. Wolfe, B. Marthi, S. J. Russell, Combined task and motion planning for mobile manipulation, in: International Conference on Automated Planning and Scheduling (ICAPS), 2010, pp. 254–258.
- [38] L. Karlsson, J. Bidot, F. Lagriffoul, A. Saffiotti, U. Hillenbrand, F. Schmidt, Combining task and path planning for a humanoid two-arm robotic system, in: ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications (TAMPRA), 2012.
- [39] K. Z. Haigh, M. M. Veloso, Interleaving planning and robot execution for asynchronous user requests, in: Autonomous agents, 1998, pp. 79–95.
- [40] L. P. Kaelbling, T. Lozano-Pérez, Hierarchical task and motion planning in the now, in: IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 1470–1477.
- [41] C. Galindo, J.-A. Fernández-Madrigal, J. González, A. Saffiotti, Robot task planning using semantic maps, Robotics and Autonomous Systems 56 (11) (2008) 955–966.

- [42] R. Petrick, D. Kraft, N. Krüger, M. Steedman, Combining Cognitive Vision, Knowledge-Level Planning with Sensing, and Execution Monitoring for Effective Robot Control, in: Workshop on Planning and Plan Execution for Real-World Systems, 2009, pp. 58–65.
- [43] J. G. Bellingham, K. Rajan, Robotics in remote and hostile environments, *Science* 318 (5853) (2007) 1098–1102.
- [44] C.-H. Cheng, M. Geisinger, H. Ruess, C. Buckl, A. Knoll, Game Solving for Industrial Automation and Control, in: IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 4367–4372.
- [45] L. de Silva, A. Pandey, R. Alami, An interface for interleaved symbolic-geometric planning and backtracking, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 232–239.
- [46] M. E. Foster, A. Gaschler, M. Giuliani, A. Isard, M. Pateraki, R. Petrick, Two People Walk Into a Bar: Dynamic Multi-Party Social Interaction with a Robot Agent, in: ACM International Conference on Multimodal Interaction (ICMI), 2012.
- [47] M. Giuliani, R. P. A. Petrick, M. E. Foster, A. Gaschler, A. Isard, M. Pateraki, M. Sigalas, Comparing Task-Based and Socially Intelligent Behaviour in a Robot Bartender, in: ACM International Conference on Multimodal Interaction (ICMI), 2013.
- [48] M. Levihn, L. P. Kaelbling, T. Lozano-Perez, M. Stilman, Foresight and reconsideration in hierarchical planning and execution, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 224–231.
- [49] D. Hadfield-Menell, E. Groshev, R. Chitnis, P. Abbeel, Modular Task and Motion Planning in Belief Space, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 4991–4998.
- [50] D. Leidner, C. Borst, Hybrid reasoning for mobile manipulation based on object knowledge, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on AI-based Robotics, 2013.
- [51] O. Khatib, A unified approach for motion and force control of robot manipulators: The operational space formulation, *IEEE Journal of Robotics and Automation* 3 (1) (1987) 43–53.

- [52] F. Lagriffoul, D. Dimitrov, A. Saffiotti, L. Karlsson, Constraint propagation on interval bounds for dealing with geometric backtracking, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 957–964.
- [53] M. R. Dogar, S. S. Srinivasa, A planning framework for non-prehensile manipulation under clutter and uncertainty, *Autonomous Robots* 33 (3) (2012) 217–236.
- [54] J. L. Barry, *Manipulation with Diverse Actions*, Ph.D. thesis, Massachusetts Institute of Technology (2013).
- [55] D. Halperin, J.-C. Latombe, R. H. Wilson, A general framework for assembly planning: The motion space approach, *Algorithmica* 26 (3-4) (2000) 577–601.
- [56] F. Gravot, S. Cambon, R. Alami, aSyMov: a planner that deals with intricate symbolic and geometric problems, *International Journal of Robotics Research (IJRR)* (2005) 100–110.
- [57] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.
- [58] J. Hoffmann, B. Nebel, The FF Planning System: Fast Plan Generation Through Heuristic Search, *Journal of Artificial Intelligence Research (JAIR)* 14 (2001) 253–302.
- [59] A. L. Blum, M. L. Furst, Fast planning through planning graph analysis, *Artificial intelligence* 90 (1) (1997) 281–300.
- [60] J. McCarthy, P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, *Readings in artificial intelligence* (1969) 431–450.
- [61] R. Reiter, The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression, *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy* 27 (1991) 359–380.
- [62] A. Ferrein, C. Fritz, G. Lakemeyer, Using golog for deliberation and team coordination in robotic soccer, *Künstliche Intelligenz (KI)* 19 (1) (2005) 24.
- [63] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – The Planning Domain Definition Language (Ver. 1.2), Tech. Rep. CVC TR-98-003, Yale Center for Computational Vision and Control (1998).

- [64] M. Helmert, The Fast Downward Planning System, *Journal of Artificial Intelligence Research (JAIR)* 26 (2006) 191–246.
- [65] C. R. Garrett, T. Lozano-Pérez, L. P. Kaelbling, FFRob: An efficient heuristic for task and motion planning, in: *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2014.
- [66] S. Srivastava, L. Riano, S. Russell, P. Abbeel, Using classical planners for tasks with continuous operators in robotics, in: *International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Planning and Robotics (PlanRob)*, 2013.
- [67] D. Nau, Y. Cao, A. Lotem, H. Muñoz-Avila, SHOP: Simple hierarchical ordered planner, in: *International joint conference on Artificial intelligence (IJCAI)*, 1999, pp. 968–973.
- [68] R. P. A. Petrick, F. Bacchus, A Knowledge-Based Approach to Planning with Incomplete Information and Sensing, in: *International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 2002, pp. 212–221.
- [69] R. P. A. Petrick, F. Bacchus, Extending the knowledge-based approach to planning with incomplete information and sensing, in: *International Conference on Automated Planning and Scheduling (ICAPS)*, 2004, pp. 2–11.
- [70] R. P. A. Petrick, A knowledge-level approach for effective acting, sensing, and planning, Ph.D. thesis, University of Toronto (2006).
- [71] R. Petrick, M. E. Foster, Planning for Social Interaction in a Robot Bartender Domain, in: *International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.
- [72] M. Garland, Quadric-based polygonal surface simplification, Ph.D. thesis, Georgia Institute of Technology (1999).
- [73] C. Ericson, *Real-time collision detection*, CRC Press, 2005.
- [74] D. P. Luebke, A developer’s survey of polygonal simplification algorithms, *Computer Graphics and Applications* 21 (3) (2001) 24–35.
- [75] D. P. Luebke, *Level of Detail for 3D Graphics*, Morgan Kaufmann, 2003.
- [76] P. Cignoni, C. Montani, R. Scopigno, A comparison of mesh simplification algorithms, *Computers & Graphics* 22 (1) (1998) 37–54.

- [77] G. Bradshaw, C. O'Sullivan, Adaptive medial-axis approximation for sphere-tree construction, *ACM Transactions on Graphics (TOG)* 23 (1) (2004) 1–26.
- [78] G. Bradshaw, Bounding volume hierarchies for level-of-detail collision handling, Ph.D. thesis, Trinity College Dublin (2002).
- [79] S. A. Ehmman, M. C. Lin, Accurate and fast proximity queries between polyhedra using convex surface decomposition, in: *Computer Graphics Forum*, Vol. 20, 2001, pp. 500–511.
- [80] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: *ACM Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 1997, pp. 209–216.
- [81] H. H. Hoppe, Progressive hulls, US Patent 6,587,104 (Jul. 1 2003).
- [82] X. Gu, S. J. Gortler, H. Hoppe, L. McMillan, B. Brown, A. Stone, Silhouette mapping, Tech. Rep. TR-1-99, Harvard University (1999).
- [83] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, J. Snyder, Silhouette clipping, in: *ACM Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 2000, pp. 327–334.
- [84] N. Platis, T. Theoharis, Progressive hulls for intersection applications, in: *Computer Graphics Forum*, Vol. 22, 2003, pp. 107–116.
- [85] N. V. Platis, *Τεχνικές πολλαπλών αναλύσεων στην απλοποίηση τριγωνικών και τετραεδρικών πλεγμάτων* (Multiresolution techniques for the simplification of triangular and tetrahedral meshes), Ph.D. thesis, University of Athens (2005).
- [86] D. Cholt, Progressive Hulls: Application on Biomedical Data, in: *Central European Seminar on Computer Graphics for students*, 2012.
- [87] C. R. Ciesla, Development of a system for the reduction of 3d polygon meshes in the field of robotics, Diplomarbeit, Technische Universität München (2007).
- [88] A. Varshney, Hierarchical geometric approximations, Ph.D. thesis, University of North Carolina at Chapel Hill (1994).
- [89] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright, Simplification envelopes, in: *ACM Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 1996, pp. 119–128.

- [90] A. Guézic, Surface simplification inside a tolerance volume, Tech. Rep. RC 20440, IBM T.J. Watson Research Center (1996).
- [91] J.-M. Lien, N. M. Amato, Approximate convex decomposition of polygons, in: Annual Symposium on Computational Geometry, 2004, pp. 17–26.
- [92] K. Mamou, F. Ghorbel, A simple and efficient approach for 3D mesh approximate convex decomposition, in: IEEE International Conference on Image Processing (ICIP), 2009, pp. 3501–3504.
- [93] B. Chazelle, D. P. Dobkin, N. Shouraboura, A. Tal, Strategies for polyhedral surface decomposition: an experimental study, *Computational Geometry* 7 (5) (1997) 327–342.
- [94] J.-M. Lien, Approximate convex decomposition and its applications, Ph.D. thesis, Texas A&M University (2006).
- [95] S. Asafi, A. Goren, D. Cohen-Or, Weak Convex Decomposition by Lines-of-sight, in: *Computer Graphics Forum*, Vol. 32, 2013, pp. 23–31.
- [96] K. Mamou, V-HACD project, <https://code.google.com/p/v-hacd/>, accessed, Aug 2015.
- [97] J. J. Kuffner, Jr., S. M. LaValle, RRT-Connect: An Efficient Approach to Single-Query Path Planning, in: IEEE International Conference on Robotics and Automation (ICRA), 2000, pp. 995–1001.
- [98] R. Smith, et al., Open dynamics engine, <http://www.ode.org/>, accessed, Aug 2015.
- [99] M. Rickert, A. Sieverling, O. Brock, Balancing Exploration and Exploitation in Sampling-Based Motion Planning, *IEEE Transactions on Robotics* 30 (6) (2014) 1305–1317.
- [100] E. G. Gilbert, D. W. Johnson, S. S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, *IEEE Journal of Robotics and Automation* 4 (2) (1988) 193–203.
- [101] P. G. Xavier, Implicit convex-hull distance of finite-screw-swept volumes, in: IEEE International Conference on Robotics and Automation (ICRA), Vol. 1, 2002, pp. 847–854.

- [102] B. Baginski, Efficient dynamic collision detection using expanded geometry models, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 3, 1997, pp. 1714–1720.
- [103] J. Schulman, A. Lee, I. Awwal, H. Bradlow, P. Abbeel, Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization, in: *Robotics: Science and Systems (RSS)*, 2013.
- [104] D. Berenson, S. Srinivasa, J. Kuffner, Task Space Regions: A Framework for Pose-Constrained Manipulation Planning, *International Journal of Robotics Research (IJRR)* 30 (12) (2011) 1435–1460.
- [105] T. Lozano-Pérez, J. Jones, E. Mazer, P. O’Donnell, W. Grimson, P. Tournassoud, A. Lanusse, Handey: A robot system that recognizes, plans, and manipulates, in: *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 4, 1987, pp. 843–849.
- [106] R. Alami, T. Siméon, J.-P. Laumond, A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps, in: *International Symposium on Robotics Research (ISRR)*, 1989.
- [107] T. Siméon, J.-P. Laumond, J. Cortés, A. Sahbani, Manipulation planning with probabilistic roadmaps, *International Journal of Robotics Research (IJRR)* 23 (7-8) (2004) 729–746.
- [108] J. Barry, K. Hsiao, L. P. Kaelbling, T. Lozano-Pérez, Manipulation with Multiple Action Types, in: *International Symposium on Experimental Robotics*, Vol. 88, 2012, pp. 531–545.
- [109] R. Alami, J.-P. Laumond, T. Siméon, Two manipulation planning algorithms, in: *Workshop on Algorithmic Foundations of Robotics (WAFR)*, 1995, pp. 109–125.
- [110] M. Stilman, J. J. Kuffner, Navigation among movable obstacles: Real-time reasoning in complex environments, *International Journal of Humanoid Robotics* 2 (04) (2005) 479–503.
- [111] M. R. Dogar, S. S. Srinivasa, Push-grasping with dexterous hands: Mechanics and a method, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 2123–2130.
- [112] J. Rutgeerts, Constraint-based task specification and estimation for sensor-based robot tasks in the presence of geometric uncertainty, Ph.D. thesis (2007).

- [113] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, H. Bruyninckx, Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty, *International Journal of Robotics Research (IJRR)* 26 (5) (2007) 433–455.
- [114] T. Lozano-Pérez, R. A. Brooks, An Approach to Automatic Robot Programming, in: M. Pickett, J. Boyse (Eds.), *Solid Modeling by Computers*, Springer, 1984, pp. 293–328.
- [115] A. Gaschler, M. Springer, M. Rickert, A. Knoll, Intuitive Robot Tasks with Augmented Reality and Virtual Obstacles, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6026–6031.
- [116] A. Ambler, R. J. Popplestone, Inferring the positions of bodies from specified spatial relationships, *Artificial Intelligence* 6 (2) (1975) 157–174.
- [117] S. Jentzsch, A. Gaschler, O. Khatib, A. Knoll, MOPL: A multi-modal path planner for generic manipulation tasks, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.
- [118] D. Berenson, S. S. Srinivasa, Probabilistically complete planning with end-effector pose constraints, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2724–2730.
- [119] L. Sentis, O. Khatib, Task-oriented control of humanoid robots through prioritization, in: *IEEE International Conference on Humanoid Robots*, 2004.
- [120] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: an open-source robot operating system, in: *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, Vol. 3, 2009, p. 5.
- [121] E. Coumans, et al., Bullet physics library, <http://bulletphysics.org/>, accessed, Aug 2015.
- [122] G. van den Bergen, et al., Solid collision detection library, <http://www.dtecta.com/>, accessed, Aug 2015.
- [123] J. Pan, S. Chitta, D. Manocha, FCL: A general purpose library for collision and proximity queries, in: *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3859–3866.

- [124] G. van den Bergen, *Collision Detection in Interactive 3D Environments*, CRC Press, 2003.
- [125] S. Nogina, *Task planning for a mobile manipulation scenario*, Master's thesis, Technische Universität München (2013).
- [126] G. J. Sussman, *A computational model of skill acquisition*, Ph.D. thesis, MIT (1973).
- [127] A. Perzylo, N. Somani, S. Profanter, M. Rickert, A. Knoll, *Toward efficient robot teach-in and semantic process descriptions for small lot sizes*, in: *Robotics: Science and Systems (RSS) Workshop on Combining AI Reasoning and Cognitive Science with Robotics*, Rome, Italy, 2015.
- [128] SMERobotics Demonstration D1, <http://www.smerobotics.org/demonstrations/d1.html>, accessed, Aug 2015.
- [129] R. Hartley, A. Zisserman, *Multiple view geometry in computer vision*, Cambridge University Press, 2003.
- [130] W. Khalil, J. Kleinfinger, *A new geometric notation for open and closed-loop robots*, in: *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 3, 1986, pp. 1174–1179.
- [131] H.-J. Siegert, S. Bocionek, *Robotik: Programmierung intelligenter Roboter*, Springer, 1996.