

TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Echtzeitsysteme und Robotik

Constraint-based Approaches for Robotic Systems: from Computer Vision to Real-Time Robot Control

Nikhil Somani

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Alin Albu-Schäffer

Prüfer der Dissertation: 1. Prof. Dr.-Ing. habil. Alois Christian Knoll
2. Prof. Dr.-Ing. Torsten Kröger

Die Dissertation wurde am 07.03.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.08.2018 angenommen.

Abstract

In this thesis, we present a framework that is designed to model and solve diverse problems using a constraint-based approach. Our motivation is to demonstrate that all robot tasks can be specified and solved as optimization problems with sets of constraints and one or multiple objective functions. We present a generic modeling approach for geometrically complex problems in the domain of intelligent robotic systems. Some of these classical problems include robot motion planning, real-time robot motion control, object detection and object tracking. We are able to develop a full constraint-based intelligent robotic system by modeling these classical problems as constraints using our framework.

Constraint models and constraint solving algorithms form the core components of our framework. Our constraint models include geometric constraints, domain-specific constraints such as robot kinematic limitations, and optimization goals such as robot manipulability. We present three constraint solving approaches: exact, iterative and hybrid. Our exact solver provides closed form and exact solutions for geometric constraints. The iterative solver supports non-linear constraints with inequalities. The hybrid solver combines the advantages of the exact and iterative solvers, by using the exact solutions as a seed for the iterative solver.

The four classical problems that form our intelligent robotic system are formulated using constraints and solved using appropriate constraint solvers from the framework.

In two robotics applications, we define robotic tasks as a set of constraints and optimization goals. To calculate the required collision-free robot motion trajectories and waypoints, we use a constrained motion planning approach based on a combination of our exact solver with a probabilistic planner, where the samples are generated in the nullspace of the task constraints. Our real-time constraint-based motion controller, based on our iterative or hybrid solver, executes this task in the presence of dynamic and environment constraints such as collisions with moving objects or optimization tasks such as posture.

We propose new approaches for classical problems in the computer vision domain such as object detection, pose estimation and tracking. Using primitive shape decompositions of object models, we pose the object detection and pose estimation problem as a constraint solving problem. We use geometric constraints from primitive shape matching in combination with our exact solver to dynamically adapt the tracking subspace of our adaptive particle filter tracker, thereby reducing the size of the tracking problem and significantly increasing its efficiency.

These applications show how constraint-based descriptions are very powerful for the underlying intelligent robotic system. A unifying framework to model and solve such diverse tasks is necessary to build a complete robotic system and apply it in practical, real-world scenarios.

An important practical application of our framework is intuitive human-robot interaction and task level robot programming. In projects related to this thesis, we have developed intuitive programming interfaces for non-expert users. The concept of task level programming, where robotic tasks are represented with reference to the involved objects and task descriptions instead of each individual action and low-level commands for the robot, is a key concept in this approach. To realize this application, we use semantically rich CAD models, robust computer vision modules, and sensor-enabled robot skills. In this thesis, we show how our constraint-based approach is used to realize the different parts of this application.

Zusammenfassung

In dieser Arbeit stellen wir ein Framework vor, das darauf ausgelegt ist, verschiedene Probleme mit einem Constraint-basierten Ansatz zu modellieren und zu lösen. Wir zeigen, wie Roboteraufgaben spezifiziert und als Optimierungsprobleme mit Constraints und einer oder mehreren Zielfunktionen gelöst werden können. Wir präsentieren einen generischen Modellierungsansatz für geometrisch komplexe Probleme. Einige dieser klassischen Probleme aus dem Bereich intelligenter Robotersysteme sind Bewegungsplanung, Echtzeit-Bewegungssteuerung, Objekterkennung und Objektverfolgung. Wir sind in der Lage, ein vollständiges auf Constraints basierendes intelligentes Robotersystem zu entwickeln, indem wir diese klassischen Probleme mit Hilfe unseres Frameworks als Constraints modellieren.

Constraint-Modelle und Constraint-Lösungsalgorithmen (Solver) bilden die Kernkomponenten unseres Frameworks. Zu unseren Constraint-Modellen gehören geometrische Constraints, domänenspezifische Constraints, z.B. kinematische Constraints von Robotern, und Optimierungsziele, wie z.B. die Manipulierbarkeit von Robotern. Wir stellen drei Lösungsansätze vor: exakt, iterativ und hybrid. Unser exakter Solver bietet eine geschlossene Form und exakte Lösungen für geometrische Randbedingungen. Der iterative Solver unterstützt nichtlineare Constraints mit Ungleichungen. Der Hybrid-Solver kombiniert die Vorteile der exakten und iterativen Solver, indem er die exakten Lösungen als Einstieg für den iterativen Solver verwendet.

Die vier klassischen Probleme, die unser intelligentes Robotersystem behandelt, werden mit Constraints formuliert und mit geeigneten Constraint-Solvern aus unserem Framework gelöst.

In zwei Anwendungen definieren wir Roboteraufgaben als eine Reihe von Constraints und Optimierungszielen. Zur Berechnung der erforderlichen kollisionsfreien Wegpunkte und Bewegungsbahnen verwenden wir einen Ansatz der Bewegungsplanung, der auf einer Kombination aus unserem exakten Solver und einem probabilistischen Planer basiert, wobei die Stichproben aus dem Nullraum der Aufgaben-Constraints gewählt werden. Unsere Constraint-basierte Echtzeit-Bewegungssteuerung basiert auf unserem iterativen oder dem hybriden Solver. Sie führt ihre Aufgabe in Gegenwart dynamischer, umgebungsbedingter Constraints, wie z.B. Kollisionen mit bewegten Objekten, aus. Weiterhin kann die Steuerung Optimierungsaufgaben übernehmen, wie z.B. eine Haltungsoptimierung des Roboters.

Wir schlagen neue Ansätze für klassische Probleme im Bereich des maschinellen Sehens vor, wie z.B. Objekterkennung, Positionsschätzung und Objektverfolgung. Mit Hilfe primitiver Formdekompositionen von Objektmodellen stellen wir Objekterkennungs- und Positionsschätzungsprobleme als ein Problem zur Lösung von geometrischen Constraints dar. Wir verwenden geometrische Constraints aus dem primitiven Formabgleich in Kombination mit

unserem exakten Solver, um den Objektverfolgungs-Teilraum unseres adaptiven Partikelfilter dynamisch anzupassen und dadurch die Komplexität der Verfolgungsaufgabe zu reduzieren und seine Effizienz signifikant zu erhöhen.

Diese Anwendungen demonstrieren die Mächtigkeit von Constraint-basierten Beschreibungen für das zugrundeliegende intelligente Robotersystem. Ein vereinheitlichendes Framework zur Modellierung und Lösung solch vielfältiger Aufgaben ist notwendig, um ein komplettes Robotersystem zu bauen und es in praktischen, realen Szenarien anzuwenden.

Eine wichtige praktische Anwendung unseres Frameworks ist die intuitive Mensch-Roboter-Interaktion und die Roboterprogrammierung auf Aufgabenebene. In Projekten im Zusammenhang mit dieser Arbeit haben wir intuitive Programmierschnittstellen für Laien entwickelt. Das Konzept der abstrakten Programmierung auf Aufgabenebene ist ein Schlüsselkonzept in diesem Ansatz, bei dem Robotik-Aufgaben mit Bezug auf die beteiligten Objekte und Aufgabenbeschreibungen anstelle von einzelnen Aktionen und Low-Level-Befehlen für den Roboter beschrieben werden. Um diese Anwendung zu realisieren, verwenden wir semantische CAD Modelle, robuste Module des maschinellen Sehens und sensorgestützte Roboterfähigkeiten. In dieser Arbeit zeigen wir, wie unser Constraint-basierter Ansatz zur Realisierung der verschiedenen Aspekte dieses intelligenten Robotersystems beiträgt.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dr. Alois Knoll and my mentor Dr. Markus Rickert for the opportunity to conduct my research in their esteemed group, and work on many interesting projects that led to this thesis. I would also like to thank Alexander Perzylo, Dr. Caixia Cai, Dr. Suraj Nair, Dr. Andre Gaschler, Yaadhav Raaaj, Stefan Profanter, Ingmar Kessler, Ahmed Rehman Ghazi, and all the other colleagues and students that I had the pleasure to work with.

Furthermore, I would also like to thank Amy Bücherl, Gertrud Eberl and Ute Lomp for their support and co-operation in all administrative tasks, and the administrative team at fortiss for creating a friendly work environment.

A special thanks to Dr. Markus Rickert and Dr. Caixia Cai for proof-reading the thesis and providing valuable feedback. Finally, I would like to thank my friends and family, especially my parents, for the support and encouragement during the work on this thesis.

Contents

List of Figures	ix
List of Tables	xiii
1 Introduction	1
1.1 Perspectives	1
1.1.1 Intuitive Human-Robot Interaction	1
1.1.2 Computer-Aided Design	2
1.1.3 Actuation	3
1.1.4 Sensing	3
1.1.5 Ubiquitous constraints	4
1.1.6 Enabling factors	5
1.2 Motivation	5
1.2.1 A humanoid example	6
1.2.2 Key ingredients	6
1.2.3 Constraints as the unifying basis	6
1.2.4 Perceiving the environment	7
1.2.5 Interacting with the environment	7
1.2.6 Semantic descriptions and logical reasoning	7
1.3 Contributions	7
1.4 Challenges	8
1.5 Applications	9
1.6 Convergence	10
1.7 Scope	11
1.8 Outline	11
I Theoretical Foundations and Models	13
2 Constraint Modeling	15
2.1 Spaces	15
2.2 Rigid Body Geometry	16
2.3 Generic Constraint Formulation	17
2.4 Geometric Constraints	17

CONTENTS

2.4.1	Geometric nullspace	19
2.4.2	Transformation submanifolds	20
2.4.3	Geometric properties of constraint nullspaces	24
2.4.4	Translation nullspace properties	24
2.4.5	Rotation nullspace properties	24
2.4.6	Context: Boundary representation of geometric entities	24
2.5	Environment and Safety Constraints	25
2.5.1	Obstacle avoidance	25
2.5.2	Robot limits	26
2.5.3	Whole body limits	26
2.5.4	Manipulability optimization	27
2.6	Discussion	27
3	Constraint Solvers	29
3.1	Related Work	29
3.2	Problem Definition	30
3.3	Motivating Example	30
3.4	Exact Solver for Geometric Constraints	30
3.4.1	Constraint processing rules	32
3.4.2	Solution synthesis	33
3.4.3	Examples	33
3.5	Iterative Constraint Solvers	38
3.5.1	Optimization in operational space	38
3.5.2	Optimization in configuration space	38
3.5.3	Task priorities and nullspaces	39
3.5.4	Parameters for iterative solver	40
3.6	Hybrid Constraint Solver	41
3.7	Constraint-based Modeling Examples	44
3.7.1	Modeling assembly tasks using geometric constraints	44
3.7.2	Constraint modeling in JSON format	44
3.7.3	3D Visualization of constraint solving	47
3.7.4	Modeling geometric constraints in an intuitive 3D GUI	47
3.8	Discussion	47
II	Applications in Robotics	49
4	Task Level Robot Programming	51
4.1	Introduction	51
4.2	Context	52
4.3	Related Work	52
4.3.1	Contributions	53
4.4	3D GUI for Task-level Robot Programming	55

4.5	Robot Skills	56
4.5.1	Welding with obstacle avoidance	56
4.5.2	Assembly of two workpieces	58
4.5.3	Cup grasping with obstacle avoidance	58
4.5.4	Tray grasping with obstacle avoidance	60
4.5.5	Pick and place with manipulability optimization	61
4.5.6	Manipulating a tray carrying an assembly	62
4.6	Applications	64
4.6.1	Gearbox assembly	64
4.6.2	Woodworking	65
4.7	Evaluation	67
4.7.1	Intuitive robot programming	68
4.7.2	Performance evaluation	70
4.8	Discussion	75
5	Constraint-based Motion Planning	79
5.1	Introduction	79
5.2	Related Work	80
5.2.1	Contribution	80
5.3	Motion Planning Approach	81
5.3.1	Sampling	81
5.3.2	Extension	84
5.3.3	Connect	84
5.4	Experiments	85
5.4.1	Moving a tray carrying an object	85
5.4.2	Seam welding with start and end point	85
5.4.3	Grasping a tray by its handle	86
5.5	Extensions	87
5.5.1	Combination with constraint-based motion controller	87
5.5.2	Including uncertainties	88
5.6	Discussion	88
III	Applications in Computer Vision	89
6	Constraint-based Object Detection and Pose Estimation	91
6.1	Introduction	91
6.2	Related Work	91
6.3	Object Detection and Pose Estimation Pipeline	93
6.4	Primitive Shape Detection	93
6.4.1	Primitive Shape hypothesis	94
6.4.2	Primitive Shape assignment	95
6.4.3	Merging and pruning of Primitive Shapes	96

CONTENTS

6.4.4	Primitive Shape Graph(PSG) representation	96
6.5	Object Instance Detection	96
6.5.1	Feature vectors for sets of Primitive Shapes	96
6.6	Object Pose Estimation	97
6.6.1	Geometric constraints from primitive shape matching	97
6.6.2	Detection of minimal and complete sets of primitives	98
6.6.3	Constraint solving for pose estimation	98
6.6.4	RANSAC based constraint solving for pose estimation	100
6.7	Evaluation	100
6.7.1	Shape-based segmentation of point clouds	100
6.7.2	Constraint-based object detection and pose estimation	101
6.8	Applications	102
6.8.1	Calculating object poses from noisy sensor data	103
6.8.2	Calculating detectability of objects from viewpoints	103
6.8.3	Reasoning about symmetrical objects	104
6.8.4	Manipulation of symmetric objects	104
6.8.5	Manufacturing uncertainties in object models	105
6.9	Discussion	105
7	Tracking using Primitive Shape Constraints (TPSC)	107
7.1	Introduction	107
7.2	Related Work	109
7.3	Primitive Shape Constraints for Tracking	110
7.3.1	Object models based on primitive shapes	110
7.3.2	Geometric model of the environment constraints	112
7.3.3	Constraint solver	112
7.4	Adaptive Particle Filter	113
7.4.1	Projection sampling	113
7.4.2	Covariance adaptation	113
7.4.3	Hypothesis verification	114
7.4.4	Complexity	115
7.5	Experiments and Evaluations	116
7.5.1	Experimental setup	116
7.5.2	Evaluation of TPSC	117
7.5.3	Application in logistics domain	118
7.6	Discussion	119
IV	Conclusion and Appendix	123
8	Conclusions and Future Work	125
8.1	Contributions	125
8.1.1	Constraint modeling and solving framework	125

8.1.2	Task level robot programming, planning and real-time control	126
8.1.3	Computer vision	126
8.1.4	Demonstrations	127
8.2	Shortcomings	127
8.2.1	Further extension of exact solver	127
8.2.2	Runtime performance of object detection	127
8.2.3	Dependence on detection of primitive shapes	128
8.3	Proposed Future Work	128
8.3.1	Convergence	128
8.3.2	Tracking in higher dimensions	129
8.3.3	Tracking and detection threads	129
8.3.4	Combined motion and force control	129
8.3.5	Constraint-based visual servoing	129
8.3.6	Extending our constraint controller for highly redundant robots	129
A	Transformation Utilities	131
A.1	Rotation defined by an axis-angle pair	131
A.2	Rotation defined by two pairs of vectors	131
B	Evaluation of Tracking with Primitive Shape Constraints	133
B.1	Tracking results for object WoodBox	133
B.2	Tracking results for object Box	133
	References	139

CONTENTS

List of Figures

1.1	Motivating example of creating object assemblies based on geometric constraints, using CAD modeling software SolidWorks	2
1.2	The kinematic structure of a robot consisting of revolute (q_1, q_2, q_4) and prismatic (q_3) joints in a closed kinematic chain.	3
1.3	Shape-based 3D object detection and pose estimation for textureless objects using an RGBD camera.	4
1.4	Motivating example of a robot carrying a wine glass on a tray. (Image from iStock.com/R_type)	5
1.5	Combining the different modules of this work to create a constraint-based robotic system architecture	10
1.6	Outline of thesis	12
2.1	Geometric constraints enforced by primitive shapes: Unconstrained axes (with x-axis in <i>red</i> , y-axis in <i>green</i> , and z-axis in <i>blue</i>) are indicated by arrows.	18
2.2	Illustration of geometric constraints and their nullspaces	21
2.3	Efficient minimum distance computation between robot links and objects	25
2.4	Constraints based on safety requirements in different human-robot interaction zones	26
2.5	Manipulability map	27
3.1	Constraint-based formulation of an assembly.	30
3.2	Pipeline for exact constraint solver.	32
3.3	Constraint combination rules for assembly.	32
3.4	Constraint combination rules for a tray grasping task.	33
3.5	Visualization of nullspaces of combinations of geometric constraints	34
3.6	Pipeline for hybrid constraint solver, involving both exact and iterative solving steps.	41
3.7	Definition of a 2-object assembly task using geometric constraints	44
3.8	GUI for loading and 3D visualization of geometric constraints between objects.	47
3.9	Constraint-based modeling example: An assembly task described using geometric constraints	48
4.1	Intuitive user-interface for definition of geometric constraints between individual shape elements of CAD models. After selecting a pair of valid surfaces, the user is presented with a preview of all valid constraints between the selected surfaces.	55
4.2	Constraint for seam welding.	57
4.3	Sample poses and task nullspaces for (a) point welding and (b) seam welding.	57

LIST OF FIGURES

4.4	Geometric constraints for the assembly of two workpieces	58
4.5	A cup grasping task expressed using geometric constraints with inequalities. The rotation and translation components in this example are not independent.	59
4.6	Sample poses and task nullspaces for the cup grasping task.	59
4.7	Manipulation of a tray: The robot task is to carry a tray that contains objects. To prevent the objects from falling, the tray must be kept upright. Allowed tolerances in rotation can be encoded as min-max values of the orientations along the respective axes.	60
4.8	Grasping a tray using a single arm robot.	60
4.9	The tray is grasped by a dual-arm robot. A set of min-max constraints can be used to express the inherent flexibility of the grasping task.	61
4.10	Constraints for grasping a cylindrical object at its rim and placing it on the table	61
4.11	Underspecified robot tasks: grasping a cylindrical object by its rim.	62
4.12	Multi-object task: manipulating a tray that carries an assembled part (Section 4.5.2)	63
4.13	Nullspace of a multi-object task: manipulating a tray that carries an assembled part	63
4.14	Assembly steps for gearbox assembly task	64
4.15	Constraints for a 4-object (2xA, 1xB, 1xC), 3-step assembly.	65
4.16	Steps and dependencies of woodworking task	66
4.17	Constraints for assembling a panel on a wooden frame.	66
4.18	Constraints for nailing a panel to a wooden frame.	67
4.19	Constraints for sawing the extra edges of a panel nailed to a wooden frame.	67
4.20	Execution of assembly task on different robotic platforms.	68
4.21	Execution of the woodworking task using a gantry robot.	69
4.22	Runtime plots for tasks in Table 4.6.	76
4.23	Execution of constraint-based tasks in a robotic workcell. CAD semantics allow these tasks to be defined in terms of geometric constraints between features of robot tools and objects, allowing intuitive robot programming.	77
4.24	Execution of constraint-based robotic tasks in the presence of environment constraints. a Cup grasping with obstacle avoidance using 6 DOF Comau RACER 7-1.4, b Cup grasping with obstacle avoidance using 7 DOF LWR, c Grasping a tray with dual arm and avoiding multiple obstacles, d Pick cylinder at rim and place with manipulability optimization.	78
5.1	Constrained Motion Planning example: Moving a tray while keeping it horizontal	86
5.2	Constrained Motion Planning example: A seam welding task	87
5.3	Constrained Motion Planning example: Grasping a tray by its handle	87
6.1	Pipeline for primitive shape based object detection from noisy sensor data	93
6.2	Pipeline for Primitive Shape Decomposition	94
6.3	Primitive Shape Decomposition example : (a) RGB channel of original Point Cloud (b) result of Primitive Shape Decomposition (c) Primitive Shape Graph representation.	94
6.4	Example of primitive shape groups including uncertainties for an industrial workpiece.	98
6.5	Primitive Shape Detection results. Cylinders are shown in red and planes are shown in blue-green.	101
6.6	RGBD Object models (a,b,c,d,e) and corresponding primitive shape decompositions (f,g,h,i,j)	102

6.7	Complete and minimal sets of primitive shapes explained through examples of an industrial workpiece.	103
6.8	Variation of primitive shape groups for different views of an object.	104
6.9	Grasping of a symmetric object: The grasp pose for the object is invariant to rotations along the symmetrical axis of the cylindrical object. Two such grasping poses are shown in this figure.	104
6.10	Handling manufacturing tolerances in production or assembly of parts in the pose estimation algorithm.	105
7.1	Constrained motion of a manipulated object: the soda can lies on a tray that is being moved by a robot.	108
7.2	RGBD Object models (a,b,c,d,e) and corresponding primitive shape decompositions (f,g,h,i,j)	111
7.3	Projection of particles to the nullspace of geometric constraints (a) 3D particle set (b) Particles projected onto a plane (c) Particles projected onto a line (d) Particles projected onto a point	113
7.4	Experimental setup with illustrations of some live tracking scenarios	116
7.5	Tracking trajectories for object JuiceBox using TPSC	117
7.6	Convergence of trackers from an initial pose, for the object Juice. The four trackers mentioned in Table 7.3 have been tested: t1 refers to “color+3D+PSC”, t2 refers to “3D+PSC”, t3 refers to “color+3D”, t4 refers to “Only 3D”.	120
7.7	Constraint-based tracking for 3 objects in our experimental setup.	121
7.8	Comparison of trajectories for object WoodBox using our live experimental setup	121
7.9	Comparison of tracking errors on synthetic data from a pre-defined trajectory for the object WoodBox.	122
8.1	Overview of contributions: our unified modeling approach, and three application domains, i.e., Task-level robot programming, Computer Vision (Model-based detection and tracking), and Robot Motion Planning and Control.	126
B.1	Comparison of trajectories for object WoodBox using our live experimental setup	134
B.2	Comparison of trajectories for object WoodBox over synthetic data with ground truth	135
B.3	Comparison of trajectories for object Box using our live experimental setup	136
B.4	Comparison of trajectories for object Box over synthetic data with ground truth	137

LIST OF FIGURES

List of Tables

2.1	Classification of constraints	18
2.2	Decomposition of separable geometric constraints into rotation and translation constraints	19
2.3	Nullspace and controlled space directions of supported geometric constraints	20
2.4	Parametric representation and point projection operations for geometric shapes	22
2.5	Parametric representation and projection for rotation submanifolds	23
2.6	Summary of supported environment/robot constraints	26
3.1	Decomposition of separable geometric constraints into basic rotation and translation constraints	35
3.2	Constraint combination rules	36
3.3	Formulation of geometric constraints for iterative solvers	37
4.1	Overall comparison of control frameworks	54
4.2	Comparison of geometric constraint solvers	54
4.3	Evaluation of time taken to program an assembly process	69
4.4	Evaluation of time taken to program a woodworking process involving the construction of a wooden wall	70
4.5	Runtime evaluation on application scenarios: solving of geometric constraints	71
4.6	Runtime evaluation on application scenarios: solving of geometric constraints with environment constraints and optimization goals	72
4.7	Convergence properties of different solvers	73
4.8	Runtime evaluation of control frameworks	74
6.1	Feature vectors for primitive shape sets	97
6.2	Summary of supported constraints between primitive shapes	97
6.3	Evaluation of object pose estimation	102
6.4	Minimal and complete sets for object pose estimation	103
7.1	Nullspace definitions for supported geometric constraints	112
7.2	Projection functions for supported geometric constraints	114
7.3	Evaluation of TPSC	118
7.4	Evaluation of TPSC in a logistics scenario	119

LIST OF TABLES

Chapter 1

Introduction

Robotics is a multi-disciplinary science, and real-world robotics applications often require contributions from several diverse fields such as control theory, planning, and computer vision. There have been many efforts to create frameworks for robotics, where these diverse fields can be combined to create practical applications. The motivation behind these frameworks is to find commonalities between these fields and exploit them to create connections and interfaces. Such observations of commonalities in terms of software structure, semantic descriptions, or underlying mathematical formulations form the key ideas behind these frameworks.

1.1 Perspectives

Creating an intelligent robotic system involves consideration of several design perspectives. These include choice of the base technologies and key concepts, along with the desired outcomes and behaviors. In the following sections, we explain some of the motivating ideas and key concepts that influence our design.

1.1.1 Intuitive Human-Robot Interaction

When intelligent robotic systems interact with a human, intuitiveness is an important consideration. This is especially relevant for robot instruction, where the human is expected to program/teach a task to the robot. The concepts of object or task centric programming can be used to increase the intuitiveness of such interactions. As an example, when a human teaches a task to a fellow human, there is a certain assumption on common knowledge of basic concepts such as colors, units, geometry etc. In case of conversations between domain experts, additional information relevant to the domain is also assumed to be known, e.g. assembly strategies, fitting tolerances. The interaction then stays largely on a higher semantic level where the focus is on the involved objects and their specific use in the task. The specification of numeric values is often restricted to a few key and rather esoteric parameters that are essential for the task in hand. Also, the focus is on the task description in terms of the involved physical objects (“what”), instead of the means to realize it (“how”), since the skills to realize them are often assumed to be common knowledge within the participants. To realize such intuitive interaction paradigms between a robot and a human, explicit and detailed models of the involved objects are essential. The amount

1. INTRODUCTION

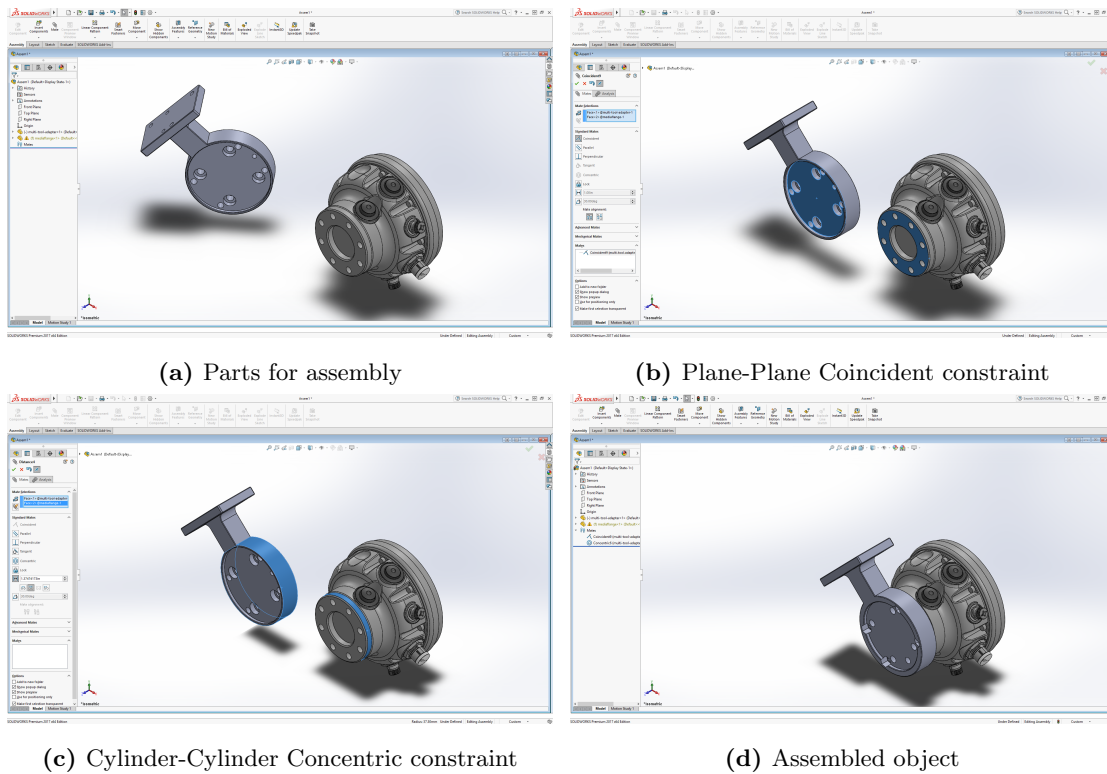


Figure 1.1: Motivating example of creating object assemblies based on geometric constraints, using CAD modeling software SolidWorks

and level of semantic information available in these models directly influences the intuitiveness of the programming approach. The object’s geometry is an important part of such models.

1.1.2 Computer-Aided Design

Computer Aided Design (CAD) is classical concept from the 1970’s, where the design focuses on the geometric properties of objects. A key idea in this paradigm is the use of a set of “primitive” geometric descriptions to construct more complex ones. Geometric objects can be created, extended and combined with each other. This paradigm has gained a lot of popularity, especially with increasing computational capabilities, resulting in more powerful CAD kernels and extensive support for 3D geometry. It is now a de-facto standard for modeling in the formal manufacturing industry. Many automation systems and manufacturing machines can directly work with data specified using CAD software. Fig. 1.1 shows an example of modeling assembly tasks using geometric constraints, based on the software SolidWorks¹. In this example, the user can import the objects to be assembled (Fig. 1.1a) and define constraints between geometric entities such as Planes (Fig. 1.1b) and Cylinders (Fig. 1.1c). The CAD software solves these constraints and generates the assembled object (Fig. 1.1d). Given the success of such approaches in providing intuitive interfaces to humans for specifying manufacturing tasks in a computer understandable format, we propose that this approach can also be used for instruction of a robotic automation system. A key factor here is the level of geometric detail that makes such systems intuitive to use. The user

¹<http://www.solidworks.com/>

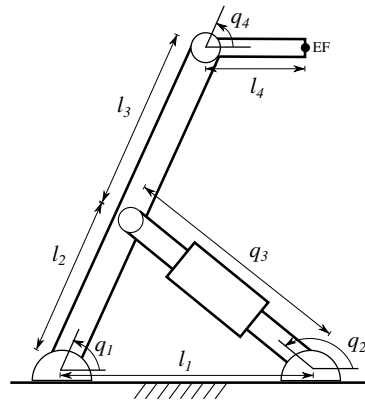


Figure 1.2: The kinematic structure of a robot consisting of revolute (q_1, q_2, q_4) and prismatic (q_3) joints in a closed kinematic chain.

interaction with geometric entities in such software is at the level of shapes such as lines, circles, planes, cylinders, etc. rather than base geometries such as meshes.

1.1.3 Actuation

When a robot interacts with its working environment, the geometrical models of the robot, the objects involved in the task and the environment are important inputs (among others) required to generate robot trajectories and closed-loop real-time controllers.

A robot is a physical entity with its own geometric description. Each link of a robot is a geometric part, and the kinematic structure of a robot can be modeled using geometric relations between its links. Consider the kinematic structure in Fig. 1.2 for a robot with 4 joints (q_{1-4}) and one end-effector (EF). The pose of the end-effector EF can be described in terms of the joint positions q_{1-4} and link lengths l_{1-4} , i.e., Forward Kinematics. Also, the joints 1,2 and 3 form a closed kinematic chain that can be modeled as a constraint between the corresponding joint positions q_{1-3} . In most practical scenarios, a desired pose for the end-effector is specified. Computation of joint positions that can achieve the required end-effector pose, i.e., Inverse Kinematics, is essentially a constraint solving problem where the desired end-effector pose is the goal or constraint and the joint positions form the optimization variables.

When interacting with the environment, tasks such as collision avoidance require computation of distances of each robot link with objects in the environment. Collision-free motion planning in the kinematics sense, and especially for tasks described in the robot's operational space, can be modeled as a constrained geometric problem.

1.1.4 Sensing

Sensors play an important role in intelligent robotic systems. They can be used at different stages, starting from intuitive programming interfaces till low-level closed-loop control. Classical computer vision topics such as object detection and tracking find direct applications in sensor-based robot motion planning and control. Vision sensors such as color and depth cameras are popular in such systems since they provide dense and feature rich information. Both 2D and 3D cameras can be used to observe the geometric structure of a scene. Especially, depth cameras directly perceive the 3D structure, i.e., the shape or



Figure 1.3: Shape-based 3D object detection and pose estimation using an RGBD camera. Shape-based approaches are particularly useful for detecting texture-less objects (a). CAD models of the detected objects are overlaid on the camera image (b). Images taken from the T-LESS dataset [1]. Copyright © 2017, IEEE.

geometry of the scene. Object detection and tracking from such sensor data, based on geometric models (2D or 3D) of the object is a classical and popular category in computer vision algorithms. Given the close integration of CAD systems with Computer-Aided-Manufacturing (CAM) systems, manufactured objects closely resemble their 3D CAD models. We focus our efforts in the computer vision domain on detecting physical instances of objects that have been modeled using CAD software (Fig. 1.3). For these objects, the geometry of the physical instance has a close correspondence with its corresponding CAD model. Our algorithms can also account for manufacturing uncertainties and tolerances introduced in CAM systems. The model-based object detection problem can be imagined as the alignment of a virtual object model with physical entities in the scene. Model-based object tracking estimates continuously and maintains a history of the pose of objects in the scene with respect to the sensor.

1.1.5 Ubiquitous constraints

A robot interacting with its physical environment essentially affects the geometric relations between objects in the environment, including itself. Controlling these geometric relations forms a part of the robot task. Essentially, each of these classical problems deal with geometric relations between real and/or virtual objects. Geometric relations can be modeled as constraints between the involved geometric entities. Consequently, these problems can be formulated using geometric constraints.

Hence, we see that Cartesian geometry is a ubiquitous mathematical theory that forms the basis of many diverse topics from different fields such as robot modeling, motion planning, object detection, tracking, etc. The geometric representations used for describing the objects, and constraint models for formulating these problems directly affects all of these fields. Increasing the level of semantic information contained in these models could potentially help improve each of these fields.

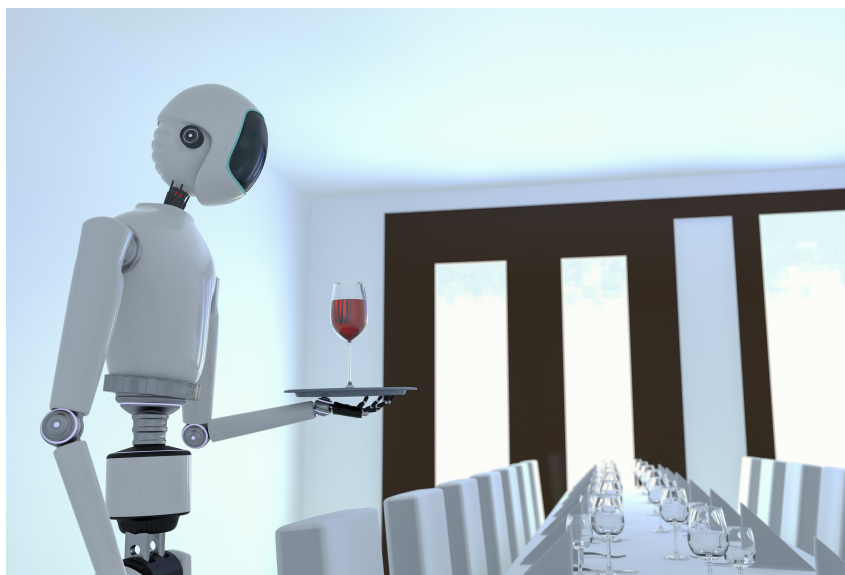


Figure 1.4: Motivating example of a robot carrying a wine glass on a tray. (Image from iStock.com/R.type)

1.1.6 Enabling factors

Some important factors that enables practical uses of our proposed approach are recent advances in computational capabilities and sensor technologies such as depth cameras. The evolution and wide acceptance of the CAD paradigm is partially attributed to increases in computational power. Due to this, the CAD kernels that form the backbone of such modeling software are becoming increasingly powerful, and the front-end interfaces are more feature-rich and responsive for a smooth and satisfactory user experience. This leads to wider acceptance of the CAD paradigm, and consequently users find such approaches also intuitive for robot programming. The advances in sensor technologies, especially affordable and accurate depth cameras has increased the relevance of 3D sensor information. Hence, computer vision algorithms based on shape or geometric information have also become more popular. An increasing demand for robotic automation technology, especially in niche applications from Small and Medium-sized Enterprises (SMEs) has also fueled research on intuitive and efficient robot programming and human-robot interaction.

1.2 Motivation

Considering all the perspectives involved in designing a robotic system, our motivation in this work is to show how it can be realized using a constraint-based approach. We explain this through an example of a common human task that should now be performed by a robot. We consider the example of a robot carrying a wine glass on a tray and transporting it towards a dining table, as illustrated in Fig. 1.4. In the following sections, we explain the complexity of this task, describe the various components and algorithms required for it, and analyze how our unified framework for modeling robotic systems using constraint-based approaches can be used to realize it.

1. INTRODUCTION

1.2.1 A humanoid example

The primary task for the robot is to transport the wine glass to the table. While performing this task, the robot has to consider several factors. Firstly, it must ensure that the tray stays upright to ensure that the wine glass doesn't fall off and the wine doesn't spill. Secondly, it needs to avoid obstacles such as the table and chairs that are on its way. This must be enforced for the whole body of the robot and not only its hand (i.e. the end-effector). Thirdly, the robot needs to continuously monitor its scene which might include moving objects. It also needs to keep the wine glass and tray in its sight to ensure their stability during motion. To do this, it maintains these objects in the field of view of the cameras that are present on its face. The robot needs to detect and track the objects of interest, i.e., the tray and wine glass. Finally, the robot may need to use its second hand in order to grasp the wine glass and place it on the table. This requires synchronization between the two hands. Note that it might not be possible to satisfy all the constraints and requirements simultaneously. Hence, priorities are defined for each constraint or requirements.

1.2.2 Key ingredients

Accomplishing this task using a robot requires contributions from several different topics in robotics. Firstly, the task itself needs to be defined in a computer-readable format. Secondly, sensors are needed to perceive the environment and corresponding computer vision algorithms to detect and track objects. Thirdly, motion planning algorithms are needed to compute a collision free path taking the robot from its current configuration to the desired configuration for the task. Finally, motion control algorithms are necessary to combine the different prioritized requirements and constraints from the task and generate real-time motion commands for the robot.

1.2.3 Constraints as the unifying basis

Our motivation in this work is to exploit the use of geometric relations as a common denominator in these topics and explore the potential benefits of using a constraint based formulation for representing them. The proposition in this thesis is to show how a constraint-based formulation of geometric relations can be adapted and re-used in several domains. In each case, the task is formulated as a set of geometric relations that constrain objects with respect to each other.

Geometric constraints are the de-facto standard approach for modeling 3D objects and creating assemblies (Fig. 1.1), and form the basis of several CAD software such as SolidWorks¹ and Catia². Our motivation is to present the user with an CAD-like interface to define robot tasks in terms of the associated objects. Referring to the example from Fig. 1.4, the task of maintaining the an upright orientation of the tray and the wine glass can be expressed using geometric constraints between CAD models of the involved objects. The task for grasping the wine glass using the second arm can also be modeled using geometric constraints between CAD models of the wine glass and the hand.

¹<http://www.solidworks.com/>

²<https://www.3ds.com/products-services/catia/>

1.2.4 Perceiving the environment

In classical computer vision modules, the object models used are often point clouds or mesh files where semantic information about geometric entities like primitive shapes is not preserved. In our framework, we show that this information of primitive shapes from full CAD models helps improve object detection and tracking algorithms. Referring to the example from Fig. 1.4, the wine glass is a symmetrical object, where the orientation along the axis that represents the stem of the wine glass is the axis of symmetry. This means that the pose of the wine glass can only be estimated in 5 DoFs, and the sixth DoF along the axis of symmetry is free. Our constraint-based object detection approach can automatically detect such symmetries, and provide both the object’s underspecified pose in the *constrained* DoFs and the pose *nullspace* that lies along this symmetrical axis. The wine glass is transported on a the tray that is placed on the robot’s hand. Using the encoders in the robot’s joints, the end-effector pose for the hand can be estimated. This information provides a prior for the pose of the tray. Further, the wine glass lies on the tray and its pose is restricted by the geometry of the tray. Using these priors represented as geometric constraints, our adaptive object tracker can automatically adapt the dimensionality and size of the tracker’s sampling subspace.

1.2.5 Interacting with the environment

Geometric constraints only represent a part of the task in each domain. We show how other aspects of each task, arising from the specific requirements of that domain, can also be represented using a constraint-based approach. We use a powerful constraint model, i.e., non-linear constraints with inequalities, so that the wide variety of constraints required for each domain can be captured easily. Referring to the example from Fig. 1.4, limitations from the robot’s kinematic model such as joint, velocity and acceleration limits can be modeled using inequality constraints. Tasks such as collision avoidance, posture optimization, and maintaining a gaze direction for the face can also be modeled using our this generic and powerful class of constraints.

1.2.6 Semantic descriptions and logical reasoning

Another motivation for constraint based definitions is that they are amenable to formal modeling and reasoning, as shown by Perzylo et al. [2, 3]. Explicitly defined mathematical formulations of constraints inherently preserve the semantic meaning of the constraint, and they can be modeled using a formal modeling language and used for high-level reasoning. This makes it easier to connect such reasoning and AI modules to low-level control components [4, 5].

1.3 Contributions

The main contribution of this thesis is the formulation of a variety of tasks from multiple domains using a constraint-based approach. We present a unifying framework for modeling these constraints and solving them. There have been individual attempts at the use of geometric constraints in different fields such as intuitive robot programming, real-time constrained motion control and constrained motion planning. In the computer vision domain, primitive shape decompositions of object models have been used for

1. INTRODUCTION

specialized object detection and tracking algorithms. However, to the best of our knowledge, a unified constraint modeling and solving framework that is applicable across these problems is a novel contribution of this thesis.

The proposed constraint solving framework is based on a composable structure where several constraints, each describing (parts of) a task or behavior, can be combined. In addition to geometric inter-relation constraints, domain specific constraints from each task/application are also handled in this framework. We present three different algorithms as part of the constraint solving framework: *exact*, *iterative* and *hybrid*.

We propose an *exact* solver for geometric constraints that is based on mathematical models of constraints and geometric properties of constraint nullspaces. Our exact solver supports non-linear geometric constraints with inequalities, and also mixed transformation manifolds, i.e., cases where the rotation and translation components of the constraints are not independent.

The *iterative* approach formulates the problems as a set of non-linear constraints with inequalities and a minimization function, and solves it using an iterative solver.

The *hybrid* solver is a combination of the *exact* and *iterative* solvers, where the transformation manifolds obtained from an exact solution of geometric constraints are represented as non-linear constraints using their distance functions. This reduces the number of geometric constraints and improves the convergence properties (solution time, repeatability, avoidance of local minima, etc.) of the iterative solver.

The overall constraint solving framework is generic and can solve any constraint that can be represented as a non-linear function with inequalities. This allows us to incorporate not only geometric constraints that can be solved exactly, but also many domain-specific constraints that may not be geometric.

We show how constraint based formulations can be used in applications from four diverse fields: real-time robot motion control, robot motion planning, object pose estimation, and model-based object tracking (explained in Section 1.5).

For each of the problem domains covered in this work, a detailed analysis of the state-of-art approaches and our specific contributions to the domain is presented in a dedicated Chapter. This comprises the *Applications* part of this thesis, i.e. Parts II and III covering Chapters 4-7.

1.4 Challenges

Progress in computing technology and the field of constraint solving has led to the availability of a large variety of solvers, that can handle many different types of problems. However, there is a major point of divergence between the constraint solving and robotics communities. The robot control domain typically deals with a relatively small set of constraints in low dimensional space. They require control algorithms that have a rather deterministic runtime and can work at high frequency control loops in the range of 1KHz. On the other hand, research in constraint solving libraries tend to focus on scaling up the performance to deal with increasingly larger problems and dimensions. Hence, there have been many attempts in robotics to develop custom solvers that are specialized to the problem types seen in the domain. Such solvers often offer better runtime performance but are not suitable for a generic constraint-based framework like ours.

From our experiments, we have observed that the most time consuming constraints in the iterative solving pipeline are the geometric constraints. By formulating the geometric constraints in a manner suitable for such solvers, and also by reducing the complexity of such geometric constraints through exactly defined constraint processing rules, we have observed that the currently available off-the-shelf constraint solvers provide sufficiently fast runtimes that are also very repeatable.

1.5 Applications

Tasks from the robotics domain are used as primary examples for our work. A majority of current robotic automation systems lack a clear distinction between *task-level* programming and robot programming. The ability to program a *task* by referring to manipulation objects and their properties, without necessarily and explicitly considering domain specific knowledge (e.g., the workspace and robot limits), is important from the viewpoint of intuitiveness. This distinction is a central concept in our approach, where our framework automatically and transparently from the user combines a task description with domain specific knowledge required for execution by a robot. Robotic tasks often do not require strictly defined goals, e.g., for a pick task on a cylindrical pipe-like object, the grasping can be done at any point on its rim. The redundancy can be even higher when these target poses are mapped to robot configurations. We propose the use of geometric inter-relational constraints to define such underspecified robot tasks. The possibility of incorporating min/max values or bounds for constraint functions significantly enhances the scope of constraint-based task definitions.

Given a constraint-based task definition, and constraint-based models of the domain-specific knowledge, our motion control framework can generate real-time motion commands for a position controlled robot. It can reactively adapt to disturbances and changes in the environment such as moving obstacles, while ensuring that the constraints are satisfied. It also supports multiple priority levels to handle situations where not all constraints can be simultaneously satisfied.

The constrained motion planning problem involves generating collision-free motion trajectories for the robot to achieve a desired goal, where the goal for the robot is defined using constraints and the robot must also follow some constraints on the path to this goal. An important concept in our work is the use of an exact geometric solver to generate task nullspaces from these constraints. These task nullspaces are essentially geometric subspaces that restrict the sampling spaces for motion planners. Our approach involves automatic generation of these geometric subspaces from constraint-based task descriptions. We present a constrained motion planning algorithm that samples in these geometric subspaces and generates trajectories of robot configurations that satisfy the path and goal constraints.

In the computer vision domain, one stream of approaches for object detection and pose estimation focuses on the use of primitive shapes in the object model. The motivation for this lies in the fact that the detection of primitive geometric shapes such as planes, cylinders, spheres, etc. is potentially faster, more repeatable and reliable than complete 3D models. Using a decomposition or approximation of the object model using primitive shapes, the object detection problem can be formulated as a shape matching problem between the primitives in the object model and the scene. Every primitive shape match acts as a geometric constraint between the object model and its instance in the scene, restricting the pose of the object in the scene. Given a set of such constraints, the object pose estimation task can be represented

1. INTRODUCTION

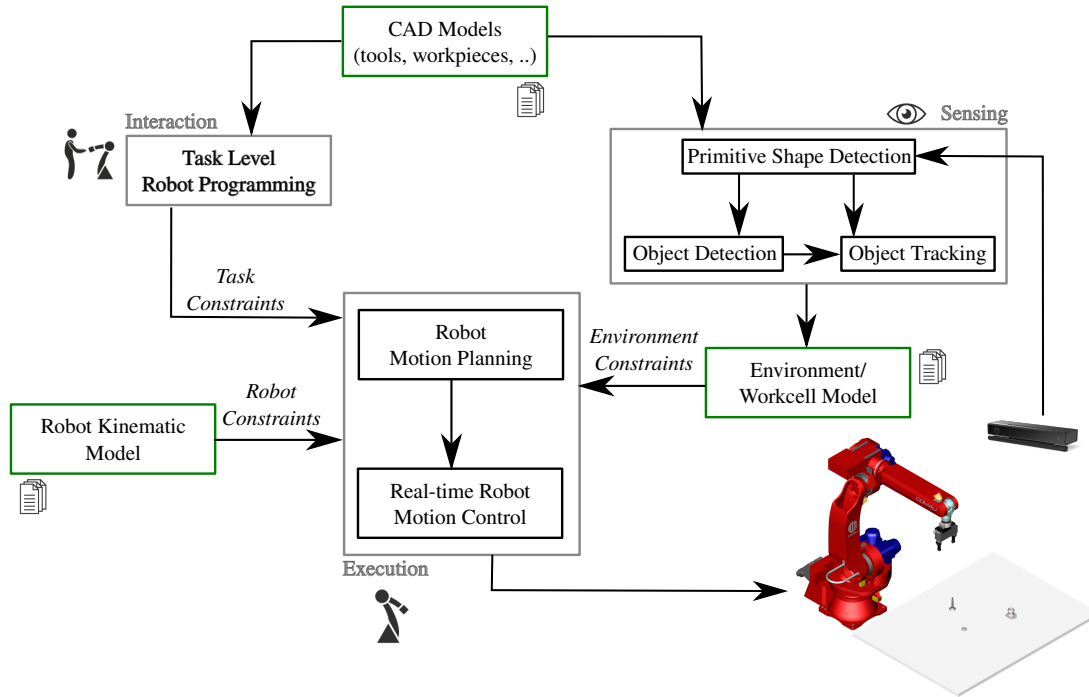


Figure 1.5: Combining the different modules of this work to create a constraint-based robotic system architecture

as a constraint solving problem. This formulation also allows the modeling of symmetrical objects and estimation of their underspecified poses.

In visual tracking, our novel geometric constraint detection and solving system helps reduce the search subspace of an adaptive Particle Filter. At every time step, it detects geometric shape constraints and associates it with the object being tracked. Using this information, it defines a new lower-dimensional search subspace for the state that lies in the nullspace of these constraints. It also generates a new set of parameters for the dynamic model of the filter, its particle count and the weights for multi-modal fusion in the likelihood modal. As a consequence, it improves the convergence robustness of the filter while reducing its computational complexity in the form of a reduced particle set.

1.6 Convergence

“Sense-Plan-Act” and “Subsumption Architectures” are things of the past in robotics, to be replaced by the “Every robot task is a constraint-optimization problem” paradigm, that links continuous, discrete and symbolic knowledge, at runtime, and all the time.

Prof. dr. ir. Herman Bruyninckx¹

The individual applications of our constraint-based approach form a part of a larger picture. Fig. 1.5 illustrates this idea, where each of these components play an important role in an intelligent robotic system architecture. There are three modules in this architecture: **interaction, sensing and execution.**

¹<https://people.mech.kuleuven.be/~bruyinck/>

CAD models provide geometric information about the different physical entities in the scene, e.g. workpieces, tools, robot structure. Intuitive interfaces use CAD models and geometric constraints to enable definition of robotic tasks at the object level. The intuitive interfaces help define the *Task Constraints*. Sensing modules ground object models in task descriptions to instances in the physical world. They exploit primitive geometric shape information from CAD models to improve the robustness and speed of object detection and tracking modules. Information from the sensing modules populates the workcell and environment and generates the *Environment Constraints*. *Robot Constraints* such as joint limits are defined by the robot’s kinematic model. Finally, the execution modules integrate constraints from these different source and realize the robot tasks on the actual hardware. Constraint-based motion planning generates desired paths for the robot. A real-time prioritized constraint-based controller uses this path as a reference, and ensures that the execution follows the constraints and their priorities in the presence of uncertainties and disturbances.

Note that application specific information such as task nullspaces, object symmetries, part manufacturing tolerances, and kinematic redundancy as well as dynamically generated data such as pose uncertainties estimated by vision algorithms and closest distances to obstacles calculated by motion planning, are shared between different modules in the system since they are all modeled using our constraint-based approach. Hence, our framework can consider all these factors simultaneously in the optimization routine.

1.7 Scope

The exact geometric solver is a core contribution of this thesis. For the iterative solving approach, we focus on the modeling of applications using constraints and making a framework amenable for use with existing non-linear solvers. An evaluation of the performance of such non-linear solvers on the applications in this thesis is also presented.

Through our experiments, we observed that solving geometric constraints takes up a significant portion of the iterative constraint solving runtime. Also, the runtime performance of iterative solvers was not sufficient for applications such as object pose estimation and tracking. We formulated and implemented the exact geometric constraint solver, which is specific to a class of constraints (geometric) and the mathematical modeling that we used for these constraints is an important contribution of this work.

We also designed a hybrid constraint solving approach that solves these geometric constraints exactly and combines them with other domain-specific constraints using an iterative solver.

1.8 Outline

This thesis is structured in three parts: **Introduction, Theory and Applications** as explained in Fig. 1.6. The first part describes the motivation leading to this work, outlines the general idea and defines the scope. The second part presents the mathematical formulations that are used throughout this work. Constraint models and constraint solving approaches defined in this part are used to develop the applications that are described in the third part.

- **Chapter 2: Constraint Modeling** – This chapter introduces the mathematical models of constraints that are used throughout this work.

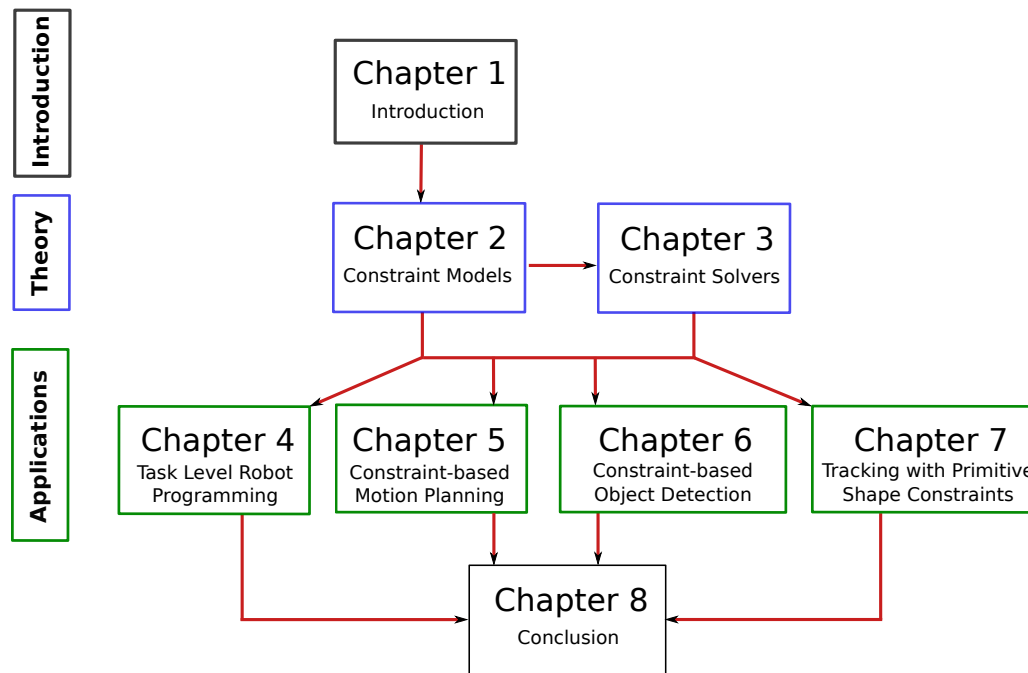


Figure 1.6: Outline of thesis

- **Chapter 3: Constraint Solvers** – This chapter describes the constraint solving framework that includes three different types of constraint solvers: exact geometric [6], iterative [7], and hybrid.
- **Chapter 4: Task Level Robot Programming** – This chapter describes the constraint-based robot control framework [7] that forms the primary application of this thesis.
- **Chapter 5: Constraint-based Object Detection and Pose Estimation** – This chapter explains how this classical problem in computer vision can be modeled and solved using the constraint-based approach [8, 9] presented in this work.
- **Chapter 6: Tracking with Primitive Shape Constraints** – This chapter shows our novel model-based 3D object tracking approach based on an adaptive Particle Filter [10] that exploits geometric properties of the object and its relation with the environment.
- **Chapter 7: Constraint-based Motion Planning** – This chapter explains our constraint-based robot motion planning approach, where the exact solver is used to generate geometric subspaces for sampling that lie in the nullspace of constraint-based robot tasks.

Part I

Theoretical Foundations and Models

Chapter 2

Constraint Modeling

In this work, we propose that constraints can be used to represent a wide variety of problems, ranging from robot control, motion planning, to object detection and tracking. The formulation of these problems using constraints is the main focus and contribution of this work. In this chapter, we describe the mathematical models that we use to represent constraints. We choose the category of non-linear constraints with inequalities for this modeling. Through several examples throughout this work, we try to demonstrate the importance and necessity of this choice.

Distinction between different types of constraints is an important concept in our approach. Robot independent task constraints depend only on the manipulation objects (e.g., assembly), manipulation constraints (e.g., grasping) depend on the object and tool, workcell constraints (e.g., collision avoidance, velocity limits) depend on the workcell and robot, and robot constraints depend only on the robot model (e.g., joint limits). To execute a task on a robot, several constraints at different levels are combined together, and solved using our constraint solving framework.

2.1 Spaces

The definition of different spaces and effective utilization of their properties is an important concept in robotic systems. In case of a robotic manipulator, there are three spaces that are most relevant to this work. First is the *Configuration Space* or the robot joint space that is directly linked to the physical drives and motors that form the robot joints. The dimensionality of this space is the same as the number of robot joints. Second is the *Operational Space* which is synonymous with the 3D Cartesian space. It is the space in which the pose of the end-effector moves. For each robot end-effector, there are 6 DoFs in the *Operational Space* comprising the 3 translation and 3 rotation DoFs. Third is the *Task Space*, which can be a subspace of the *Operational Space* or the *Configuration Space* depending on the task definition. This space defines the valid or desired region where the task requirements are fulfilled. In this thesis, we also call this space as the *Task NullSpace*.

Each of these spaces can be used to control the robot, and the choice depends on their properties. At the low levels of the real-time robot controller, the drive and motors are directly controlled. The *Configuration Space* is the most appropriate space for this level. This is also the reason why most robot control interfaces support control in the *Configuration Space*. The *Operational Space* has a direct

2. CONSTRAINT MODELING

connection to the 3D Cartesian world. It can be more intuitive for a human operator to program a robot in this space rather than the *Configuration Space*. The *Task Space* is often used for constraint-based control applications. There are different ways in classical literature for defining this space and designing a controller to operate the robot within its limits. Our work also focuses on the use of *Task Space* control. More specifically, we focus on geometric aspects of the *Task Space*. At the position level, the mapping from *Configuration Space* to *Operational Space* is called the Forward Kinematics (FK), and the reverse is called the Inverse Kinematics (IK). At the velocity level, the forward mapping is called the Robot Jacobian (\mathbf{J}). At the velocity level, the mapping from the *Configuration Space* to the *Task Space* is called the Task Jacobian (\mathbf{J}_e).

In the computer vision domain, the space of transformations between object models and their instances in a visual scene is termed the *Operational Space*. For a single asymmetrical rigid object, this space is 6 dimensional. The dimensionality can be lower for symmetrical objects. For articulated objects, this space can have higher DoFs depending on the number of rigid bodies comprising the object and their relative configuration. The *Detection Space* or *Tracking Space* of an object is a subspace of the *Operational Space*. Constraints from Primitive Shape matches or additional information about the environment affect the dimensionality of these spaces. The performance of a detection or tracking algorithm is often directly and heavily influenced by this space. A key idea of this work is to exploit these constraints to reduce the dimensionality or size of the *Detection Space* or *Tracking Space* compared to the *Operational Space*. We also present detection and tracking algorithms that can operate in these subspaces instead of the *Operational Space*.

2.2 Rigid Body Geometry

A rigid body is defined as a shape in \mathbb{R}^n such that the distance between any two points on it is constant over time ¹. A rigid transformation in the \mathbb{R}^n space belongs to the special Euclidean Space $\mathbf{T} \in SE(n)$. It is a mapping in the \mathbb{R}^n space:

$$\mathbf{T}(\mathbf{p}) = \mathbf{R}\mathbf{p} + \mathbf{t}, \quad (2.1)$$

where $\mathbf{R} \in SO(n)$ denotes the rotation and $\mathbf{t} \in \mathbb{R}^n$ denotes the translation.

In this work, we consider only rigid transformations in 3-dimensional space, i.e., $\mathbf{T} \in SE(3)$, $\mathbf{R} \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$. We represent the rotation matrix either as Axis-Angles ² or Quaternions ³. The transformation matrix is then represented as a 7-element vector: $[t_x, t_y, t_z, aa_x, aa_y, aa_z, aa_\theta]$ or $[t_x, t_y, t_z, q_x, q_y, q_z, q_w]$ where $[t_x, t_y, t_z]$ is the 3-D translation, $[aa_x, aa_y, aa_z, aa_\theta]$ is the Axis-Angle notation for \mathbb{R} and $[q_x, q_y, q_z, q_w]$ is the Quaternion notation for \mathbb{R} . This choice is based on our observation from our experiments that Quaternions and Axis Angles are better suited for non-linear optimization routines compared to Euler Angles. Quaternions have the additional advantage of a more continuous representation of the rotation space, and often yield better convergence properties for non-linear solvers as compared to Axis Angles.

¹https://en.wikipedia.org/wiki/Rigid_body

²https://en.wikipedia.org/wiki/Axis-angle_representation

³https://en.wikipedia.org/wiki/Quaternion#Quaternions_and_the_geometry_of_R3

2.3 Generic Constraint Formulation

There are several properties related to a generic constraint formulation: the space of its optimization variables \mathbb{R}^n , the operational space in which the constraint values lie \mathbb{R}^m , the mathematical model of this constraint $C \in \mathbb{C}$, the subspace in operational space $\mathcal{M}_m \in \mathbb{R}^m$ that forms the nullspace of the constraint, and the subspace of its optimization variables $\mathcal{M}_n \in \mathbb{R}^n$ that forms its constraint nullspace.

In context of rigid geometric constraints, the space of optimization variables and the operational space are the same, i.e., the 3D rigid transformation space $SE(n)$. A constraint $C \in \mathbb{C}$ on the transformation T between two objects restricts the space of this transformation from $SE(3)$ to a smaller subspace $\mathcal{M} \in SE(3)$, which is also called the nullspace of this constraint.

The space of optimization variables \mathbb{R}^n is not necessarily the Euclidean space. In the robotics context, this could also be the space of robot joint configurations. The generic constraint formulation still holds in this space. The constraints $C \in \mathbb{C}$ and the corresponding nullspace \mathcal{M} depend on the specific task. We study mathematical modeling techniques to represent constraints relevant to our application scenarios.

Several works in classical and recent literature have successfully used constraint categories such as linear with equality, linear with inequality, or non-linear with equality. This choice is justified by the specific requirements of their use-case, thereby allowing them to design custom constraint solving approaches. However, in a more generic, cross-domain framework such as ours, we choose the most powerful model of non-linear constraints with inequalities that subsumes all the other mentioned types. We justify this choice in each of our application domains (robot control, object detection, visual tracking, and motion planning) by providing examples of common tasks that cannot be represented easily using simpler constraint categories. Through evaluations, we also show that despite using a more complex constraint model, our solver formulations are efficient.

The general form of the non-linear inequality constraint used in this work is shown in (2.2). For better convergence, the function $g_i(\mathbf{x})$ is a metric or pseudo-metric in the space \mathbf{x} .

$$\text{lb}(g_i(\mathbf{x})) \leq g_i(\mathbf{x}) \leq \text{ub}(g_i(\mathbf{x})), \quad i = 1, \dots, m. \quad (2.2)$$

In the following sections, we derive the expressions for the constraint function $g_i(\mathbf{x})$ and its boundaries $[\text{lb}(g_i(\mathbf{x})), \text{ub}(g_i(\mathbf{x}))]$ for the constraints used in our applications.

We study two different types of constraints: geometric and robot environment. While ubiquitous geometric constraints are the most important contribution of this work, environment constraints for robotic tasks are important for real-world robotic applications. Section 2.4 explains the models used for geometric constraints. Section 2.5 explains the constraints that arise from the robot model or the environment. Note how all these examples can be represented using non-linear constraints with inequalities.

2.4 Geometric Constraints

Given a fixed and a constrained shape, a geometric constraint adds restrictions to the relative pose of the constrained shape with respect to the fixed shape. Some examples of these constraints are illustrated in Fig. 2.1. While the definition of constraint functions depends on the involved geometric entities, their bounds also depend on user-specified minimum and maximum values for the constraints. The constraint

2. CONSTRAINT MODELING

Table 2.1: Classification of constraints

Constraint Type	Example	Solution nullspace
Geometric, separable R,t	Pl-Pl Coinc	Plane
Geometric, inequality, separable R,t	Pl-Pl Dist Min-Max	Box
Geometric, non-separable R,t	Cyl-L Dist, Tangent	1 DoF sub-manifold
Geometric, inequality, non-separable R,t	Cyl-L Dist Min-Max, Tangent	2 DoF sub-manifold
non-linear	manipulability maximization	-
non-linear, inequality	collision avoidance	configuration sub-space

¹ Pl = Plane, L = Line, Pt = Point, Cyl = Cylinder, Coinc = Coincident, Dist = Distance

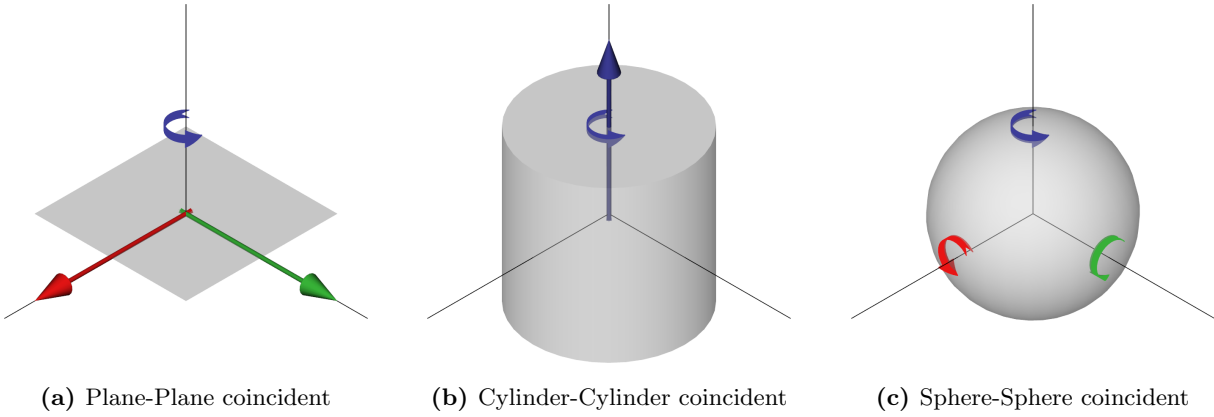


Figure 2.1: Geometric constraints enforced by primitive shapes: Unconstrained axes (with x-axis in *red*, y-axis in *green*, and z-axis in *blue*) are indicated by arrows.

function defines the shape of the constraint nullspace, while the bounds define the boundaries of this shape.

Geometric constraints can be categorized in several ways. Firstly, depending on equations representing the constraints between objects, they can be classified as linear or non-linear. Secondly, the rotation and translation components of the constraints may or may not be independent. This independence greatly simplifies the constraint solving algorithm [11]. However, as we demonstrate in this work, this limitation restricts the expressive power of constraint-based descriptions. In Section 3.4, we propose a solution for cases where the rotation and translation constraints depend on each other. Table 2.1 lists the different types of geometric constraints based on two criteria: Inequality support and separability of rotation and translation components. This classification is important for the solving step in Section 3.4.

Table 2.2 lists some of the geometric constraints that are supported by our system and are relevant to our use-cases. In case of separable constraints, the rotation and translation components of these constraints are defined.

As explained in the following Sections, we define several properties and functions for geometric constraints that are useful for building applications around constraint-based problem descriptions:

- Constraint decomposition rules that express the constraint as a combination of simpler constraints, or as rotation and translation constraints in case the constraint is separable (Table 2.2).
- Controlled space directions (\mathcal{S}) and null space directions (\mathcal{N}) (Table 2.3).
- Projection function that projects a point \mathbf{u} onto the geometric nullspace: $\text{geomProject}(\mathbf{u})$ (Tables 2.4 and 2.5).
- Closest distance of a point \mathbf{u} to the geometric nullspace: $\text{dist}(\mathbf{u}) = \text{distance}(\mathbf{u}, \text{geomProject}(\mathbf{u}))$, where distance is a distance metric between two rigid transformations in the \mathbb{R}^3 space.
- Parametric representation of nullspace (Tables 2.4 and 2.5).
- Constraint functions for iterative solvers: g (Table 3.3)

2.4.1 Geometric nullspace

A geometric constraint restricts the possible space of transformations between a fixed and constrained shape. The free/permissible space of such transformations form the constraint *nullspace*, while the restricted space forms the *constrained* space. An illustration of these spaces for some simple spaces is shown in Fig. 2.1. As an example, the Plane-Plane coincident constraint in Fig. 2.1 (a) restricts the rotation along the x (red) and y (green) axes, and translation along the z axis (blue). The nullspace consists of translation along the x (red) and y (green) axes, and rotation along the (blue) axis. Table 2.3 provides the relevant definitions for some common constraints.

While a nullspace in general is a manifold in N-dimensional space, a geometric nullspace has some special properties that relate to its geometric origin. In some cases, it can also be visualized as a shape with a boundary representation. We denote the geometric nullspace as a manifold \mathcal{M} , which has several

Table 2.2: Decomposition of separable geometric constraints into rotation and translation constraints

Fixed	Constr.	Constraint	Rotation	Translation
L ₁	Pt ₂	Dist (d_{\min}, d_{\max})	–	$d_{\min} \leq d(L_1, Pt_2) \leq d_{\max}$
L ₁	L ₂	Dist (d_{\min}, d_{\max})	$\angle(a_1, a_2) = 0^2$	$d_{\min} \leq d(L_1, Pt_2) \leq d_{\max}$
L ₁	L ₂	Angle (a_{\min}, a_{\max})	$a_{\min} \leq \angle(a_1, a_2) \leq a_{\max}$	–
Pl ₁	Pt ₂	Dist (d_{\min}, d_{\max})	–	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$
Pl ₁	L ₂	Dist (d_{\min}, d_{\max})	$\angle(n_1, n_2) = \pi/2$	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$
Pl ₁	L ₂	Angle (a_{\min}, a_{\max})	$\pi/2 - a_{\min} \leq \angle(n_1, n_2) \leq \pi/2 - a_{\max}$	–
Pl ₁	Pl ₂	Dist (d_{\min}, d_{\max})	$\angle(n_1, n_2) = 0$	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$
Pl ₁	Pl ₂	Angle (a_{\min}, a_{\max})	$a_{\min} \leq \angle(n_1, n_2) \leq a_{\max}$	–
L ₁	Pl ₂	Tangent	non-separable	non-separable

¹ Pl = Plane, L = Line, Pt = Point, Coinc = Coincident, Dist = Distance, || = Parallel

² skew or intersecting lines are not considered

2. CONSTRAINT MODELING

Table 2.3: Nullspace and controlled space directions of supported geometric constraints

Fixed	Constrained	Constraint (C)	Controlled Space Directions (S)	Null Space Directions (N)
Pl ₁	Pl ₂	Dist (d)	$S_R : [n_{1\perp 1}; n_{1\perp 2}]$ $S_t : [n_1]$	$N_R : [n_1]$ $N_t : [n_{1\perp 1}; n_{1\perp 2}]$
Pl ₁	Pl ₂		$S_R : [n_{1\perp 1}; n_{1\perp 2}]$ $S_t : []$	$N_R : [n_1]$ $N_t : \mathbf{1}$
L ₁	L ₂		$S_R : [n_{1\perp 1}; n_{1\perp 2}]$ $S_t : [n_{1\perp 1}; n_{1\perp 2}]$	$N_R : [n_1]$ $N_t : [n_1]$
L ₁	L ₂	Coinc	$S_R : [n_{1\perp 1}; n_{1\perp 2}]$ $S_t : [n_{1\perp 1}; n_{1\perp 2}]$	$N_R : [n_1]$ $N_t : [n_1]$
L ₁	Pt ₂	Coinc	$S_R : []$ $S_t : [n_{1\perp 1}; n_{1\perp 2}]$	$N_R : \mathbf{1}$ $N_t : [n_1]$
L ₁	Pt ₂	Dist (d)	$S_R : [n_{1\perp 1}; n_{1\perp 2}]$ $S_t : []$	$N_R : [n_1]$ $N_t : [n_1]$

¹ Pl = Plane, L = Line, Pt = Point, Coinc = Coincident, Dist = Distance, || = Parallel

properties that are explained in the following Sections: parametric equation, projection function, distance function, separation rules, etc.

2.4.2 Transformation submanifolds

Submanifolds for geometric constraints whose rotation and translation elements are decoupled were defined for relative positioning tasks in [11]. We add definitions for some additional submanifolds (e.g., Box, Parallelepiped, ThickCylinder) that are required for inequality constraints.

Some mathematical definitions and properties of translation submanifolds, such as their parametric equations and projection functions, are summarized in Table 2.4. The equations and properties of rotation submanifolds are very similar to the ones defined in [11]. Apart from the exact geometric constraint solver, a closed form definition of these functions is very important for the visual tracking (Chapter 7) and motion planning (Chapter 5) applications.

We also define full transformation submanifolds, where each DoF in the submanifold nullspace can be a mixture of rotation and translation DoFs. An example of such a manifold is shown in Fig. 2.2i. In this example, the rotation and translation elements depend on each other and cannot be separated. For any point lying on Line₂, the translation is restricted along the Cylinder submanifold. If the translation along the Cylinder submanifold is considered as a DoF, the rotation is fully defined by the two perpendicularity constraints. Conversely, if the rotation along axis Line₁ is considered to be the DoF, the translation is fully defined. Hence, the nullspace of this constraint is a 1 DoF manifold where the rotation and translation cannot be independently controlled at the same time.

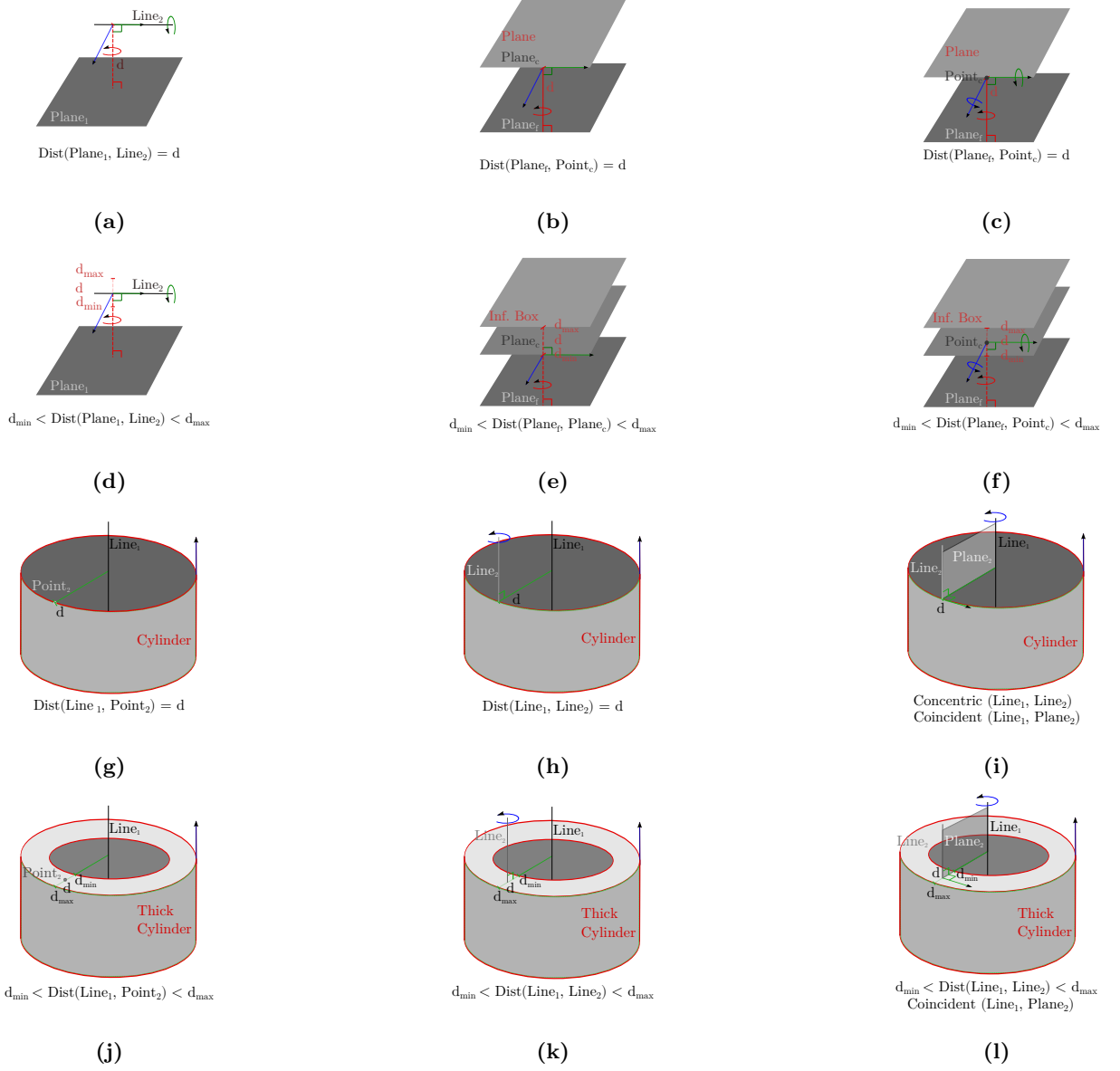


Figure 2.2: Illustration of geometric constraints and their nullspaces. (a) The Plane-Line distance constraint (red) defines the distance of Line_2 from Plane_1 . The translation nullspace of this constraint is indicated by the green and blue arrows. The rotation nullspace is around the axes marked by red and green arrows. (g) The Line-Point distance constraint (red) defines the distance of Point_2 from Line_1 . The translation nullspace of this constraint is the *Cylinder* (red). The rotation is completely unconstrained. (h) The Line-Line distance constraint (red) defines the distance of a point on Line_2 from Line_1 . The axis-axis-parallel constraint (green) defines that the axes of the two lines should be parallel. The translation nullspace of this constraint is the *Cylinder* (red). The rotational nullspace is shown using the blue arrow. (i) Example of a set of constraints, where the rotation and translation are non-separable. The Line-Point distance constraint (red) defines the distance of the center point of the Plane from the Line. The Line-Circle-Tangent constraint (green) defines that the normal direction of the Plane should be tangential to the Circle shown in red. In a combination of these constraints, the rotation and translation elements are non-separable. The composite nullspace of this transformation manifold is the rotation around the Line (shown in blue). (j)–(l) The constraints can also be extended with inequalities to define the minimum and maximum distances.

Table 2.4: Parametric representation and point projection operations for geometric shapes

Shape/submanifold	Parametric representation	Projection of Point: $\text{geomProject}(\mathbf{u})$
\mathbb{R}^3	$\mathbf{q}(s_1, s_2, s_3) = s_1 \hat{\mathbf{d}}_1 + s_2 \hat{\mathbf{d}}_2 + s_3 \hat{\mathbf{d}}_3$	\mathbf{u}
Point (\mathbf{p})	$\mathbf{q} = \mathbf{p}$	\mathbf{p}
Line $(\mathbf{p}, \hat{\mathbf{a}})$	$\mathbf{q}(s) = \mathbf{p} + s \hat{\mathbf{a}}$	$\mathbf{p} + ((\mathbf{u} - \mathbf{p}) \cdot \hat{\mathbf{a}}) \hat{\mathbf{a}}$
Circle $(\mathbf{p}, \hat{\mathbf{n}}, r)$	$\mathbf{q}(\alpha) = \mathbf{p} + r(c_\alpha \hat{\mathbf{n}}_{\perp 1} + s_\alpha \hat{\mathbf{n}}_{\perp 1})$	$\mathbf{p} + r[\text{geomProject}_{\text{Plane}(\mathbf{p}, \hat{\mathbf{n}})}(\mathbf{u}) - \mathbf{p}]^1$
Plane $(\mathbf{p}, \hat{\mathbf{n}})$	$\mathbf{q}(s_1, s_2) = \mathbf{p} + s_1 \hat{\mathbf{n}}_{\perp 1} + s_2 \hat{\mathbf{n}}_{\perp 2}$	$\mathbf{u} + ((\mathbf{p} - \mathbf{u}) \cdot \hat{\mathbf{n}}) \hat{\mathbf{n}}$
Sphere (\mathbf{p}, r)	$\mathbf{q}(\alpha, \beta) = \mathbf{p} + r(c_\alpha c_\beta \hat{\mathbf{d}}_1 + s_\alpha s_\beta \hat{\mathbf{d}}_2 + c_\beta \hat{\mathbf{d}}_3)$	$\mathbf{p} + r[\mathbf{u} - \mathbf{p}]^1$
Cylinder $(\mathbf{p}, \hat{\mathbf{a}}, r, h)$	$\mathbf{q}(s, \alpha) = \mathbf{p} + s \hat{\mathbf{a}} + r(c_\alpha \hat{\mathbf{a}}_{\perp 1} + s_\alpha \hat{\mathbf{a}}_{\perp 1})$	$\text{geomProject}_{\text{Plane}(\mathbf{p}, \hat{\mathbf{n}})}(\mathbf{u}) + r[\mathbf{u} - \text{geomProject}_{\text{Plane}(\mathbf{p}, \hat{\mathbf{n}})}(\mathbf{u})]^1$
Box $(\mathbf{p}, \mathbf{d}, \mathbf{l})$	$\mathbf{q}(s_1, s_2, s_3) = \mathbf{p} + s_1 \hat{\mathbf{d}}_1 + s_2 \hat{\mathbf{d}}_2 + s_3 \hat{\mathbf{d}}_3, s_i \in [-l_i/2, l_i/2]$	
Parallelepiped $(\mathbf{p}, \mathbf{n}, \mathbf{l})$	$\mathbf{q}(s_1, s_2, s_3) = \mathbf{p} + s_1 \hat{\mathbf{n}}_1 + s_2 \hat{\mathbf{n}}_2 + s_3 \hat{\mathbf{n}}_3, s_i \in [-l_i/2, l_i/2]$	
ThickCylinder $(\mathbf{p}, \hat{\mathbf{a}}, r, h, w)$	$\mathbf{q}(s_1, s_2, \alpha) = \mathbf{p} + s_1 \hat{\mathbf{a}} + (r + s_2)(c_\alpha \hat{\mathbf{a}}_{\perp 1} + s_\alpha \hat{\mathbf{a}}_{\perp 1})$	

¹ vectors enclosed in [] are considered normalized

Table 2.5: Parametric representation and projection for rotation submanifolds

Submanifold	DoF	# Solutions	Parametric representation $R(\phi)$	Projection of Rotation R_i
\mathbb{R}^3	3	1	$R(\phi_1, \phi_2, \phi_3)$	R_i
OneParallel	1	1	$R_{AA}(\hat{\mathbf{v}}_f, \phi)$	$R_{AA}(\hat{\mathbf{w}}_R, \alpha_R)R_i$
OneAngle (θ)	2	1	$R_{AA}(\hat{\mathbf{v}}_f, \phi_1)R_{AA}(\hat{\mathbf{w}}, \alpha - \theta)R_{AA}(\hat{\mathbf{v}}_c, \phi_2)$	$R_{AA}(\hat{\mathbf{w}}_R, \alpha_R - \theta)R_i$
OneAngleMinMax ($\theta_{min}, \theta_{max}$)	2	1	$R_{AA}(\hat{\mathbf{v}}_f, \phi_1)R_{AA}(\hat{\mathbf{w}}, \text{rDiff}(\alpha, [\theta_{min}, \theta_{max}]))R_{AA}(\hat{\mathbf{v}}_c, \phi_2)$	$R_{AA}(\hat{\mathbf{w}}_R, \text{rDiff}(\alpha_R, [\theta_{min}, \theta_{max}]))R_i$
TwoParallel	0	1	$R_{vv}((\hat{\mathbf{v}}_{c1}, \hat{\mathbf{v}}_{c2}) \rightarrow (\hat{\mathbf{v}}_{f1}, \hat{\mathbf{v}}_{f2}))$	$R_{vv}((R_i\hat{\mathbf{v}}_{c1}, R_i\hat{\mathbf{v}}_{c2}) \rightarrow (\hat{\mathbf{v}}_{f1}, \hat{\mathbf{v}}_{f2}))$
TwoAngle	1	2		
TwoAngleMinMax	1	2		
ThreeAngle	0	≤ 8		
ThreeAngleMinMax	0	≤ 8		

¹ \mathbf{v}_f = Fixed vector, \mathbf{v}_c = Constrained vector, $\hat{\mathbf{w}} = \hat{\mathbf{v}}_f \times \hat{\mathbf{v}}_c$, $\alpha = \angle(\hat{\mathbf{v}}_f, \hat{\mathbf{v}}_c)$, $\alpha_R = \angle(\hat{\mathbf{v}}_f, R_i\hat{\mathbf{v}}_c)$, $\hat{\mathbf{w}}_R = \hat{\mathbf{v}}_f \times R_i\hat{\mathbf{v}}_c$, R_i = Input rotation matrix, $R_{AA}(\mathbf{a}, \theta)$ = Rotation matrix defined by Axis \mathbf{a} and angle θ (Section A.1), $\text{rDiff}(a, [b_{min}, b_{max}]) = \{0 \text{ if } b_{min} < a < b_{max}, b_{max} - a \text{ if } b_{max} > a, a - b_{min} \text{ if } a < b_{min}\}$, R_{vv} is the rotation matrix computed using two pairs of corresponding vectors $((\hat{\mathbf{a}}, \hat{\mathbf{b}}) \rightarrow (\hat{\mathbf{d}}, \hat{\mathbf{e}}))$ (Section A.2)

2.4.3 Geometric properties of constraint nullspaces

The nullspace of geometric constraints can be expressed in the form of transformation submanifolds, which have a geometric meaning (with parametric equations and projection functions) of their own. Hence, the nullspace of such geometric constraints can often (but not always) be exported as CAD files and visualized using 3D rendering software. Fig. 2.2 shows several examples of constraint nullspaces and their geometric interpretations, including all categories of geometric constraints defined in Table 2.1. Such geometric representations or visualizations can be derived easily in some cases, but can become very complicated for certain types of constraints, especially in case of constraints with inequalities.

2.4.4 Translation nullspace properties

For each geometric shape, we define properties and functions that are necessary for constraint solvers. The parametric equation for a shape allows sampling 3D points that lie inside the shape. For the 3D Cartesian space \mathbb{R}^3 , the parametric equation of a point \mathbf{q} depends on 3 free parameters (s_1, s_2, s_3) , and is given by $\mathbf{q}(s_1, s_2, s_3) = s_1\hat{\mathbf{d}}_1 + s_2\hat{\mathbf{d}}_2 + s_3\hat{\mathbf{d}}_3$, where $(\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2, \hat{\mathbf{d}}_3)$ are basis vectors in the space \mathbb{R}^3 . This space has 3 degrees of freedom, which is the same as the size of its parameter space.

Similarly, for a plane defined by a point (\mathbf{p}) on it and a normal direction ($\hat{\mathbf{n}}$), a point in the parametric space \mathbf{q} depends on 2 free parameters (s_1, s_2) . The point can be represented as $\mathbf{q}(s_1, s_2) = \mathbf{p} + s_1\hat{\mathbf{n}}_{\perp 1} + s_2\hat{\mathbf{n}}_{\perp 2}$, where $\hat{\mathbf{n}}_{\perp 1}$ and $\hat{\mathbf{n}}_{\perp 2}$ are perpendicular to $\hat{\mathbf{n}}$.

A projection function (`geomProject`) calculates the closest point on the shape for a given input 3D point. Table 2.4 defines these properties for the translation submanifolds that are relevant to the examples in this work.

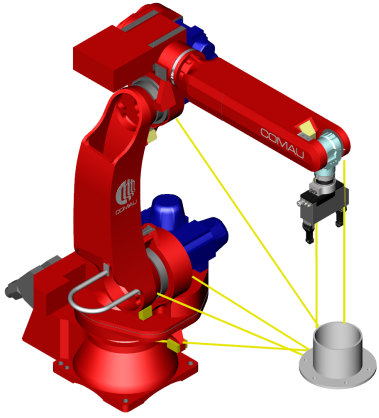
2.4.5 Rotation nullspace properties

Similar to translations, properties such as parametric equation and projection functions are defined for rotation submanifolds. Table 2.4 defines these properties for the rotation submanifolds that are relevant to the examples in this work.

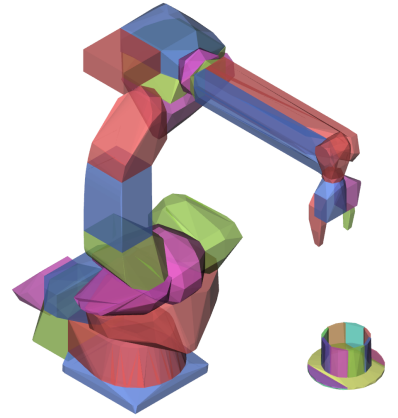
The space of 3D rotations $SE(3)$ can be parametrically defined using 3 parameters (ϕ_1, ϕ_2, ϕ_3) . Note that the projection of an input rotations on a submanifold might not be unique, e.g. the `ThreeAngle` constraint can have 8 possible solutions. In such cases, the projected rotation that is closest to the input rotation is selected.

2.4.6 Context: Boundary representation of geometric entities

We use a boundary representation (BREP) [2] for all geometric entities in the robotic system. This representation decomposes each object into semantically meaningful primitive shapes (e.g., planes, cylinders). For each of these shape types, Table 2.4 lists the parametric equation used to define the geometric nullspace, and a projection function to project a point onto the shape. Conversely, the nullspaces of geometric constraints can be expressed using transformation submanifolds, which can be modeled using BREP.



(a) Shortest distances between object and robot links.



(b) Bounding convex decomposition with convex segments shown in different colors.

Figure 2.3: Efficient minimum distance computation between robot links and objects using Gilbert/Johnson/Keerthi algorithm [12].

2.5 Environment and Safety Constraints

In addition to direct geometric relations, the robot controller needs to incorporate safety requirements arising from the robot model (e.g., joint limits) and from the workcell/environment (e.g., collision avoidance). Besides these, there are additional limitations on the robot’s speed and acceleration when a human is present in the robot’s workspace to ensure safe collaboration. These constraints are summarized in Table 2.6.

2.5.1 Obstacle avoidance

Avoiding collisions with other physical entities in the environment while executing a task is an important and nearly essential requirement for all robotic tasks. In our approach for realizing this, minimum distances are efficiently computed between each link of the robot and the object to be avoided using the Gilbert/Johnson/Keerthi (GJK) algorithm [12]. Each distance computation provides the minimum distance d_{ij} , and the points \mathbf{p}_i and \mathbf{p}_j on the robot link B_i and obstacle O_j respectively (Fig. 2.3(a)). If the distance d is below a threshold value d_{thresh} , the robot motion is constrained in a way that it does not move closer to the object, see (2.3).

$$d_{\text{thresh}} \leq d_{ij} = \|\mathbf{p}_i - \mathbf{p}_j\|, \forall B_i, O_j \quad (2.3)$$

Computation of distances between general meshes is often computationally expensive. Convex meshes can simplify this to a large extent. In contrast to Schulman et al. [13] who create an approximate convex decomposition of all robots and objects, we perform a bounding convex decomposition [14, 15] (Fig. 2.3(b)). A bounding convex decomposition encloses the original geometry and contains fewer vertices, effectively allowing safe and efficient collision avoidance. On the convex decomposition, shortest distances are calculated with the GJK algorithm [12].

2. CONSTRAINT MODELING

Table 2.6: Summary of supported environment/robot constraints

Fixed	Constr.	Constraint (i)	Constr. Function (g_i)	lb	ub
Object O_j	Link B_i	Distance (d_{\min})	$\ (\mathbf{p}_i - \mathbf{p}_j)\ $	d_{\min}	–
–	Robot	Joint Angles ($\mathbf{q}_{\min}, \mathbf{q}_{\max}$)	$[\mathbf{q}]$	\mathbf{q}_{\min}	\mathbf{q}_{\max}
–	Robot	Joint Velocities ($\dot{\mathbf{q}}_{\min}, \dot{\mathbf{q}}_{\max}$)	$[\dot{\mathbf{q}}]$	$\dot{\mathbf{q}}_{\min}$	$\dot{\mathbf{q}}_{\max}$
–	Robot	Cart. Velocity ($\dot{\mathbf{x}}_{\min}, \dot{\mathbf{x}}_{\max}$)	$[\dot{\mathbf{x}}]$	$\dot{\mathbf{x}}_{\min}$	$\dot{\mathbf{x}}_{\max}$
–	Robot	Manipulability	$1/(1 + \sqrt{\det(\mathbf{J}\mathbf{J}^T)})$	–	–

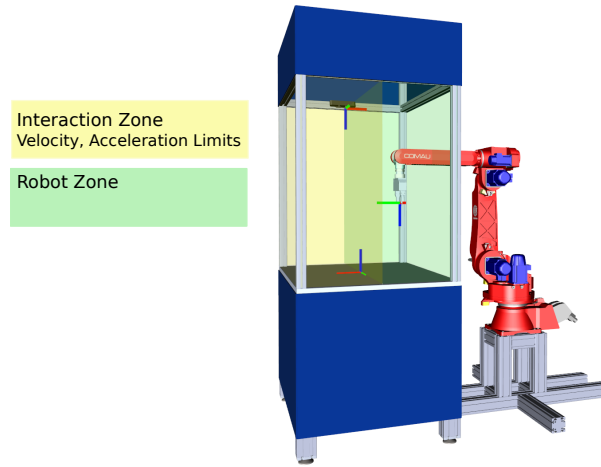


Figure 2.4: Constraints based on safety requirements in different human-robot interaction zones

2.5.2 Robot limits

Limitation of the robot kinematic structure, e.g., joint position/velocity/acceleration limits, can be modeled as inequality constraints (Table 2.6).

$$\begin{aligned}
 \mathbf{q}_{\min} &\leq \mathbf{q} \leq \mathbf{q}_{\max} \\
 \dot{\mathbf{q}}_{\min} &\leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\max} \\
 \dot{\mathbf{x}}_{\min} &\leq \dot{\mathbf{x}} \leq \dot{\mathbf{x}}_{\max}
 \end{aligned} \tag{2.4}$$

2.5.3 Whole body limits

Limitations on the operational position/velocity/acceleration of each robot link or operational element can also be modeled as inequality constraints (Table 2.6). Certain regions of the robot's workspace can be distinguished, e.g., based on a risk assessment of a human-robot interaction scenario. For instance, different velocity and acceleration limits can be defined within the linked geometric volumes, i.e., the yellow and green cuboids (Fig. 2.4). The green cuboid is defined as the robot's zone, where the human is not allowed or expected to operate. Hence, the whole body velocity limits for this region are higher than that for the interaction zone (yellow cuboid).

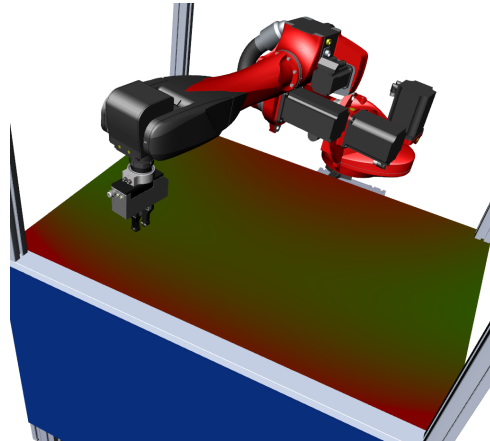


Figure 2.5: Manipulability map

2.5.4 Manipulability optimization

In addition to hard limits that arise from the robot or environment model, there are some criteria that need to be minimized or maximized. One such measure is the manipulability measure, introduced by Yoshikawa [16]. It indicates the feasibility of velocities in different Cartesian directions from a given robot pose. It can be used as a quality measure for robot poses, and the optimization goal is to maximize this measure (Table 2.6).

Equation (2.5) defines a unit sphere in joint velocity space, which can be projected into Cartesian space using the pseudo-inverse of the robot Jacobian $\mathbf{J}^\#$ (2.6). This can be simplified to (2.7), which then represents the space of feasible Cartesian velocities.

$$\dot{\mathbf{q}}^T \dot{\mathbf{q}} = 1 \quad (2.5)$$

$$(\mathbf{J}^\# \dot{\mathbf{x}})^T \mathbf{J}^\# \dot{\mathbf{x}} = 1 \quad (2.6)$$

$$\dot{\mathbf{x}}^T (\mathbf{J}\mathbf{J}^T)^{-1} \dot{\mathbf{x}} = 1 \quad (2.7)$$

This can also be understood in terms of the eigenvalues and eigenvectors of $\mathbf{J}\mathbf{J}^T$, where the eigenvectors represent the axis directions, and square-roots of the eigenvalues the lengths of the axes. Yoshikawa's manipulability measure $\sqrt{\det(\mathbf{J}\mathbf{J}^T)}$ specifies the volume of the ellipsoid and is maximized in isotropic configurations (where the lengths of the axes are equal).

Fig. 2.5 presents an example of this optimization for a scenario where the robot pose is constrained to points on the table. The color code on the table indicates the manipulability measure at different positions. Green values indicate higher manipulability, while red ones indicate low manipulability. As expected, points that lie close to the joint limits of the robot have lower manipulability, while those lying towards the center of the joint ranges are higher.

2.6 Discussion

In this Chapter, we presented our mathematical models for constraints. We covered a wide variety of constraints, ranging from geometric constraints between primitive shapes to domain-specific constraints

2. CONSTRAINT MODELING

such as robot joint limits, and optimizations goals such as manipulability maximization. We presented detailed models of constraints including properties such as constrained-space and nullspace, projection and distance functions.

In the next Chapter, we present constraint solving approaches that can generate target configurations that conform to a given set of constraints. The mathematical models presented in this chapter form the basis for the formulation of optimization problems for the solvers in Chapter 3.

Chapter 3

Constraint Solvers

This chapter focuses on our constraint solving framework. Three different approaches for constraint solving have been presented. Each solving approach has its own unique features, and the choice of solver depends on the requirements of the applications. This choice is put into context and explained in each of the application chapters.

3.1 Related Work

There are two major categories geometric constraint solving approaches [17],[18]: iterative and exact. Iterative approaches model the constraints using cost functions and pose the solving process as an optimization problem. Exact solvers use geometric properties of the involved entities to create constraint simplification and combination rules, along with a mapping of constraints to geometric nullspaces. Iterative approaches are generally easier to model, and can represent constraints which can be difficult to model geometrically. However, they inherit problems from the non-linear optimization domain, such as a lack of convergence guarantee with different starting values, local minima, numerical stability issues when using derivatives, and a non-deterministic runtime. Exact solvers are designed to give repeatable and guaranteed results from any starting pose, with a deterministic runtime which can be significantly faster than iterative approaches [7].

In this work, we propose and analyze three different approaches: a purely iterative solver which is very generic [7], an exact solver specialized for geometric constraints [6], and a hybrid solver. The hybrid solver first uses an exact geometric solver to compute the geometric nullspace. An iterative solver then optimizes over the robot joint positions such that the end-effector pose lies in this geometric nullspace. From our experiments, it can be observed that by partially processing the geometric constraints using the exact solver, the convergence properties of the subsequent iterative solver are significantly better. This is due to the fact that the effective dimensionality of the iterative solver's search space is reduced. Also, the robot pose obtained from projections to the geometric nullspace acts as a good starting seed value for the iterative solver, thereby reducing chances of divergence.

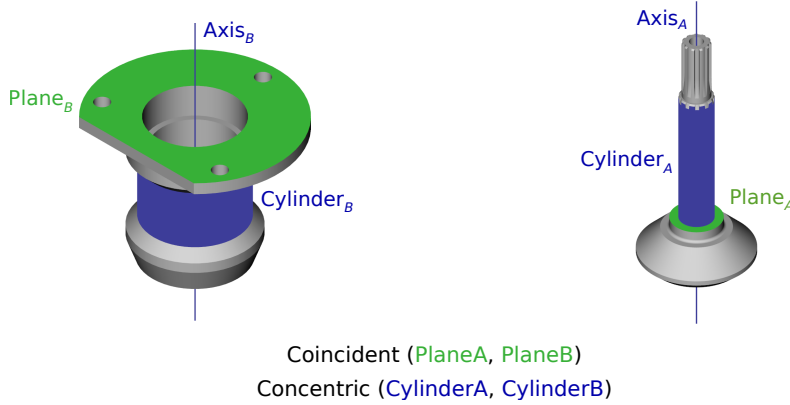


Figure 3.1: Constraint-based formulation of an assembly task. Object A containing the primitive shapes Axis_A , Cylinder_A and Plane_A is fixed. Object B containing the primitive shapes Axis_B , Cylinder_B and Plane_B is constrained with respect to A .

3.2 Problem Definition

The generic constraint solving approach, defined in (3.1), performs optimization in the space of an optimization variable \mathbf{v} . It requires an optimization function $f(\mathbf{v})$ along with constraints $\mathbf{C}(\mathbf{v})$. The goal of this optimization is the minimization of the optimization function $f(\mathbf{v})$, while ensuring the constraints $\mathbf{C}(\mathbf{v})$ are satisfied. Priorities between constraints can also be handled using this generic equation in multiple steps. At each step, the high priority constraints are set as as constraints ($\mathbf{C}(\mathbf{v})$) and lower priority constraints are optimized in the least squared sense by adding them to the optimization function $f(\mathbf{v})$ (Section 3.5.3).

$$\begin{aligned} & \arg \min_{\mathbf{v}} && f(\mathbf{v}) \\ & \text{subject to} && \mathbf{C}_i(\mathbf{v}) \in \mathbb{C} \quad i = 1, \dots, m. \end{aligned} \tag{3.1}$$

3.3 Motivating Example

We choose the assembly task presented in Fig. 3.1 as an example that will be used to explain our different constraint solving approaches. This task is defined using two geometric constraints between primitive shapes of two objects. One constraint is a `PlanePlaneCoincident` constraint between Plane_B and Plane_A . The second constraint is a `CylinderCylinderConcentric` constraint between Cylinder_B and Cylinder_A . Object A is fixed and object B is constrained.

3.4 Exact Solver for Geometric Constraints

The geometric constraints solving problem can be defined as an approach to calculate a relative pose between two objects (in general), based on a set of constraints between their geometries. Hence, given a set of geometric constraints \mathbb{C} between a fixed object O_f and a constrained object O_c and an initial transformation T_f^c , the goal is to compute a transformation $T_f^{ms(c)}$ that is the projection of T_f^c in the

```

Input: Constraints  $\mathbb{C}$ , initial constrained object pose  $T_f^c$ 
Output: Solved constrained object pose  $T_f^{ns(c)}$ 
solvable  $\leftarrow$  false
foreach  $C_i \in \mathbb{C}$  do // separate constraints into R,t (Table 3.1)
    if  $\exists$  separation( $C_i$ ) then
        |  $\mathbb{C} \leftarrow (\mathbb{C} - C_i) \cup$  separation( $C_i$ )
        end
    end
foreach pair  $C_i, C_j \in \mathbb{C}$  do // reduce constraints using combination rules (Table 3.2)
    if  $\exists$  combinationRule( $C_i \cup C_j$ ) then
        |  $\mathbb{C} \leftarrow (\mathbb{C} - (C_i \cup C_j)) \cup$  combinationRule( $C_i \cup C_j$ )
        end
    if  $\exists$  manifoldMap( $\mathbb{C}$ ) then // map constraint set to manifold (Table 3.1)
        | solvable  $\leftarrow$  true
        | break
    end
end
if solvable then
    manifold  $\leftarrow$  manifoldMap( $\mathbb{C}$ )
    if separable(manifold) then // project on R, t manifolds (Table 2.4)
        |  $R_f^{ns(c)} \leftarrow$  geomProject( $R_f^c$ ) * ( $R_f^c$ )T
        |  $t_f^{ns(c)} \leftarrow$  geomProject( $t_f^c$ ) - (geomProject( $R_f^c$ ) * ( $R_f^c$ )T *  $t_f^c$ )
        |  $T_f^{ns(c)} \leftarrow$  ( $R_f^{ns(c)}, t_f^{ns(c)}$ )
    else // project on transformation manifold
        |  $T_f^{ns(c)} \leftarrow$  geomProject( $T_f^c$ )
    end
end

```

Algorithm 1: Exact constraint solving approach: separation is a set of constraint decomposition rules defined in Table 3.1. combinationRule is a set of constraint combination rules, some of which have been defined in Table 3.2 and illustrated in Figs. 3.3 and 3.4. manifoldMap is the mapping of constraints to transformation manifolds, as defined (not exhaustively) in Table 3.1. geomProject is a projection function for each translation and rotation transformation submanifold.

nullspace of the geometric constraints \mathbb{C} :

$$T_f^{ns(c)} = \text{projection}(\mathbb{C}, T_f^c). \quad (3.2)$$

In case of iterative solvers, each constraint ($C_i \in \mathbb{C}$) defines a cost function $\text{CF}(C_i, T_f^c)$. The constraint solving problem is then expressed as an optimization problem:

$$T_f^{ns(c)} = \arg \min_{C_i \in \mathbb{C}} \text{CF}(C_i, T_f^c). \quad (3.3)$$

In case of exact geometric solvers, the set of constraints \mathbb{C} is first decomposed into simpler constraints (and in case of separable constraints, their rotation and translation components) using the rules defined

3. CONSTRAINT SOLVERS

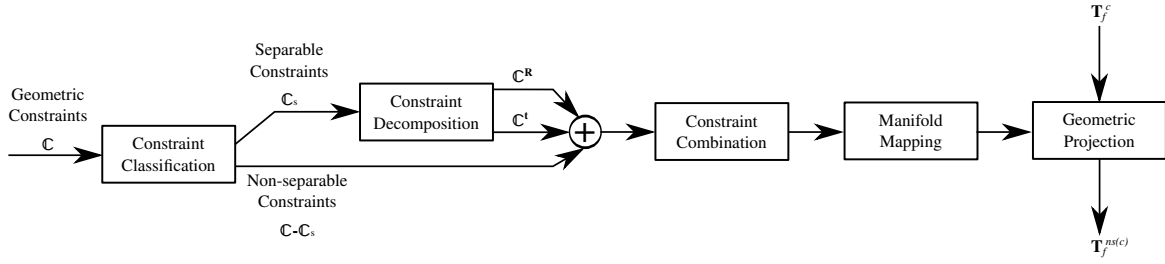


Figure 3.2: Pipeline for exact constraint solver.

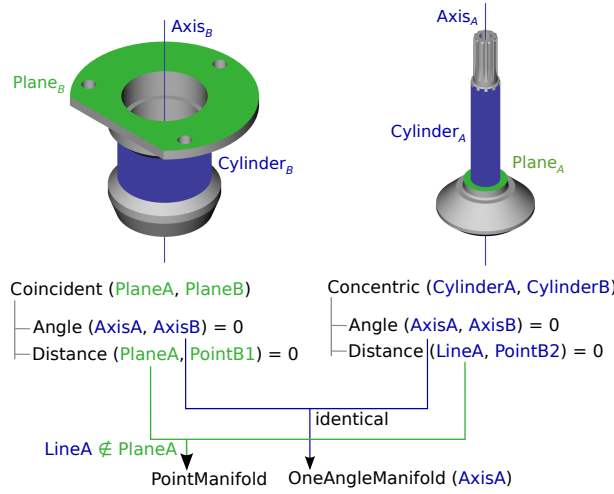


Figure 3.3: Example of constraint combination rules for assembly.

in Table 3.1. These constraints are then combined according to specified constraint combination rules (examples are shown in Table 3.2). These constraint processing step are important to simplify and reduce the set of input constraints to a set that can be easily mapped to submanifolds. Some of the rules for mapping sets of constraints to transformation submanifolds are shown in Table 3.1. Given the transformation submanifold, the transformation T_f^c can be projected to $T_f^{ns(c)}$ using the submanifold's projection function (geomProject defined in Table 2.4).

The algorithm for the exact geometric solver is explained in Algorithm 1. The solving pipeline is illustrated in Fig. 3.2 and the most important steps, i.e., the constraint processing rules and solution synthesis, are explained in the following sections.

3.4.1 Constraint processing rules

The simplification and combination of constraints is an important step in the constraint solving process. By separating complex constraints into a combination of simpler ones, the number of constraint combination scenarios and rules can be reduced. The combination rules for a constraint pair also perform a consistency check by analyzing their geometric properties and determining whether they are compatible or redundant. By combining constraints in this way, the number of mappings needed from constraint sets to transformation manifolds can also be reduced.

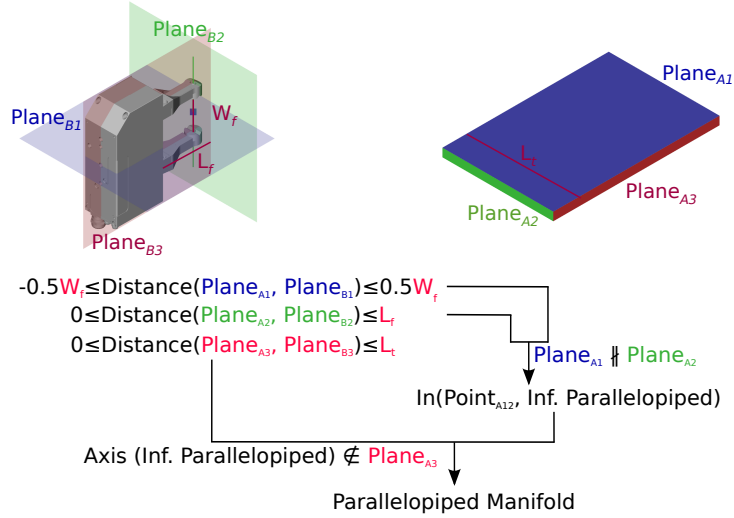


Figure 3.4: Constraint combination rules for a task with inequality constraints (tray grasping).

Some constraint simplification and combination rules have been presented in [11]. We created additional rules in this work to enable the combination of constraints with inequalities (Table 3.2). Also, for constraints where the rotation and translation sub-manifolds are dependent on each other (Table 3.1), full transformation manifolds are required to represent their nullspace (Section 2.4.2).

After simplification, the constraints are recursively combined till a set of constraints that can be directly mapped to a transformation manifold is obtained. Some examples of these mappings between constraints and transformation manifolds is described in Table 3.1.

3.4.2 Solution synthesis

Once the transformation manifold has been determined, the closest operational pose for the robot is calculated by projecting the current pose onto the transformation manifold. The projection functions for rotation and translation manifolds are defined in Tables 2.4 and 2.5. The projection functions find the closest pose in the least-squares sense. Projection functions that work on mixed transformation manifolds calculate this based on a weighted distance in rotation and translation spaces.

3.4.3 Examples

Fig. 3.3 presents an example of constraint processing rules for the task that we presented in Section 3.3. Both constraints from this example are separable into rotation and translation components (Table 3.1). The $\text{Coincident}(\text{Plane}_A, \text{Plane}_B)$ constraint can be decomposed into two constraints: rotation constraint $\text{Coincident}(\text{Axis}_A, \text{Axis}_B)$ and translation constraint $\text{Distance}(\text{Plane}_A, \text{Point}_{B1}) = 0$, where Point_{B1} is a point on the constrained object B that lies on Plane_B . Similarly, the $\text{Concentric}(\text{Cylinder}_A, \text{Cylinder}_B)$ constraint can be decomposed into two constraints: rotation constraint $\text{Coincident}(\text{Axis}_A, \text{Axis}_B)$ and translation constraint $\text{Distance}(\text{Line}_A, \text{Point}_{B2}) = 0$, where Line_A is the axis of fixed Cylinder_A and Point_{B2} is a point on the constrained object B that lies on the axis of Cylinder_B .

The rotation components from both constraints are identical, and can be mapped to a OneParallel rotation manifold.

3. CONSTRAINT SOLVERS

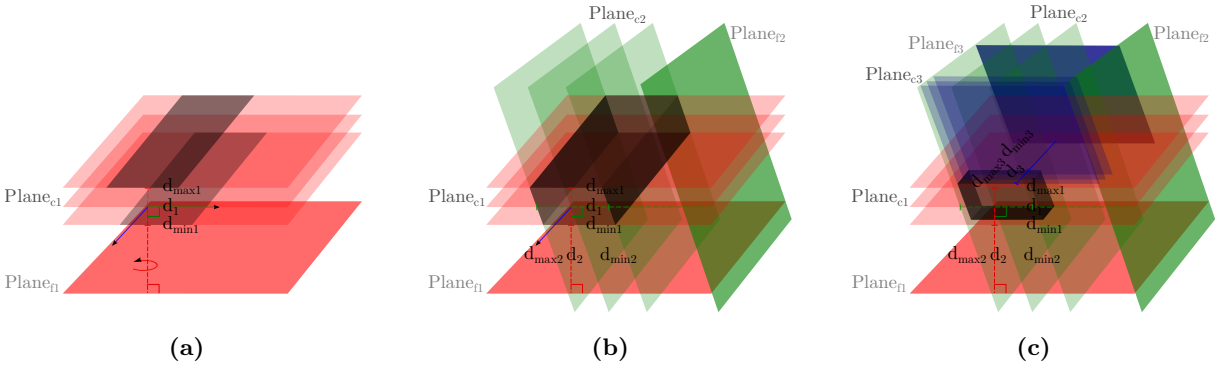


Figure 3.5: Visualization of nullspaces of combinations of geometric constraints: (a) One plane-plane-distance-min-max constraints generates an infinite Box as the nullspace, where the constrained axis of the Box lies along the normal direction of the fixed plane. (b) Two plane-plane-distance-min-max constraints generate an infinite Parallelepiped as the nullspace, where the infinite axis of the Parallelepiped lies along the cross product of the normal directions of the fixed planes. (c) Three plane-plane-distance-min-max constraints generate a finite Parallelepiped as the nullspace.

The translation components can be combined according to the case where $Line_A$ is not contained in the $Plane_A$ (see row 4 of Table 3.2). The resulting translation constraint is a $Coincident(Point_A, Point_{B12})$ constraint, where $Point_A$ is the point of intersection of $Plane_A$ and $Line_A$ and $Point_{B12}$ is the point of intersection of constrained geometries $Plane_B$ and $Axis_B$ of $Cylinder_B$. This $Coincident(Point_A, Point_{B12})$ constraint can be mapped to a Point translation manifold according to Table 3.1.

Fig. 3.4 presents an example of constraint processing rules for a task that requires inequality constraints (Section 4.5.4). This task comprises three $PlanePlaneDistanceMinMax$ constraints. In the absence of inequalities, these constraints would result in a fully constrained operational position. As shown in Fig. 3.5 and Table 3.1, the geometric nullspace of this combination of constraints is a Finite Parallelepiped. In this example, the three sets of planes are perpendicular to each other. Hence, the corresponding nullspace is a special case of the Finite Parallelepiped, i.e., a Finite Cube.

More examples of constraint processing rules based on robotics use-cases are presented in Section 4.5.

Table 3.1: Decomposition of separable geometric constraints into basic rotation and translation constraints

Fixed	Constr.	Constraint	Rotation	Translation	Transformation Manifold (Translation, Rotation)
L ₁	Pt ₂	Dist (d_{\min}, d_{\max})	–	$d_{\min} \leq d(L_1, Pt_2) \leq d_{\max}$	ThickCylinder, \mathbb{R}^3 (Fig. 2.2j)
L ₁	L ₂	Dist (d_{\min}, d_{\max})	$\angle(a_1, a_2) = 0^2$	$d_{\min} \leq d(L_1, Pt_2) \leq d_{\max}$	ThickCylinder, OneParallel (Fig. 2.2k)
L ₁	L ₂	Angle (a_{\min}, a_{\max})	$a_{\min} \leq \angle(a_1, a_2) \leq a_{\max}$	–	\mathbb{R}^3 , OneAngleMinMax
Pl ₁	P ₂	Dist (d_{\min}, d_{\max})	–	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$	Inf. Box, \mathbb{R}^3
Pl ₁	L ₂	Dist (d_{\min}, d_{\max})	$\angle(n_1, n_2) = \pi/2$	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$	Inf. Box, OneAngle (Fig. 2.2d)
Pl ₁	L ₂	Angle (a_{\min}, a_{\max})	$\pi/2 - a_{\min} \leq \angle(n_1, n_2) \leq \pi/2 - a_{\max}$	–	\mathbb{R}^3 , OneAngleMinMax
Pl ₁	Pl ₂	Dist (d_{\min}, d_{\max})	$\angle(n_1, n_2) = 0$	$d_{\min} \leq d(Pl_1, Pt_2) \leq d_{\max}$	Pl, OneParallel
Pl ₁	Pl ₂	Angle (a_{\min}, a_{\max})	$a_{\min} \leq \angle(n_1, n_2) \leq a_{\max}$	–	\mathbb{R}^3 , OneAngleMinMax
L ₁	Pl ₂	Tangent	non-separable	non-separable	Fig. 2.2l

¹ Pl = Plane, L = Line, Pt = Point, Coinc = Coincident, Dist = Distance, || = Parallel² skew or intersecting lines are not considered

Table 3.2: Constraint combination rules¹

Constraint 1	Constraint 2	Combination condition	Combined constraint
$\text{Coinc}(Pl_1^f, Pt_1^c)$	$\text{Coinc}(Pl_2^f, Pt_2^c)$	$Pl_1^f \# Pl_2^f, Pt_1^c = Pt_2^c$	$\text{Coinc}(L_{12}^f, Pt_1^c)$
$\text{Coinc}(Pl_1^f, Pt_1^c)$	$\text{Coinc}(L_2^f, Pt_2^c)$	$Pt_{12}^f = L_2^f \cup Pl_1^f, Pt_1^c = Pt_2^c$	$\text{Coinc}(Pt_{12}^f, Pt_1^c)$
$\text{Dist}(Pl_1^f, Pt_1^c) \in [d_{\min 1}, d_{\max 1}]$	$\text{Dist}(Pl_2^f, Pt_2^c) \in [d_{\min 2}, d_{\max 2}]$	$Pl_1^f \# Pl_2^f, Pt_1^c = Pt_2^c$	$\text{In}(\text{Inf.Ppd}_{12}^f, Pt_1^c)$
$\text{Dist}(Pl_3^f, Pt_3^c) \in [d_{\min 3}, d_{\max 3}]$	$\text{In}(\text{Inf.Ppd}_2^f, Pt_2^c)$	$\text{Ppd}_2^f \notin Pl_3^f, Pt_3^c = Pt_2^c$	$\text{In}(\text{Ppd}_{12}^f, Pt_3^c)$
$\text{Dist}(Pl_1^f, Pt_1^c) \in [d_{\min 1}, d_{\max 1}]$	$\text{Coinc}(L_2^f, Pt_2^c)$	$L_2^f \# Pl_1^f, Pt_1^c = Pt_2^c$	$\text{Coinc}(\text{LS}_{12}^f, Pt_1^c)$
$\text{Coinc}(Pl_1^f, Pt_1^c)$	$\text{Dist}(L_2^f, Pt_2^c) \in [d_{\min 2}, d_{\max 2}]$	$L_2^f \# Pl_1^f, Pt_1^c = Pt_2^c$	$\text{Coinc}(\text{El}_{12}^f, Pt_1^c)$
$\text{Coinc}(Pl_1^f, Pt_1^c)$	$\text{Dist}(L_2^f, Pt_2^c) \in [d_{\min 2}, d_{\max 2}]$	$L_2^f \perp Pl_1^f, Pt_1^c = Pt_2^c$	$\text{Coinc}(\text{Circ}_{12}^f, Pt_1^c)$
$\text{Dist}(Pl_1^f, Pt_1^c) \in [d_{\min 1}, d_{\max 1}]$	$\text{Dist}(L_2^f, Pt_2^c) \in [d_{\min 2}, d_{\max 2}]$	$L_2^f \perp Pl_1^f, Pt_1^c = Pt_2^c$	$\text{In}(\text{Cyl}_{12}^f, Pt_1^c)$

¹ Pl = Plane, L = Line, LS = Line Segment, Pt = Point, Ppd = Parallelepiped, Inf.Ppd = Infinite Parallelepiped, El = Ellipse, Circ = Circle, Cyl = Cylinder, Coinc=Coincident, Dist=Distance

Table 3.3: Formulation of geometric constraints for iterative solvers

Fixed	Constrained	Constraint (i)	Constraint Function (g_i)	Lower Bound (lb)	Upper Bound (ub)
Pt ₁	Pt ₂	Dist (d_{\min}, d_{\max})	$[\mathbf{p}_{21}^T \mathbf{p}_{21}]$	$[d_{\min}^2]$	$[d_{\max}^2]$
Pl ₁	Pl ₂	Dist Angle ($d_{\min}, d_{\max}, \theta_{\min}, \theta_{\max}$)	$[\mathbf{n}_1^T \mathbf{p}_{21}; \mathbf{n}_2^T \mathbf{n}_1]$	$[d_{\min}; \cos(\theta_{\max})]$	$[d_{\max}; \cos(\theta_{\min})]$
L ₁	L ₂	Dist (d_{\min}, d_{\max})	$[\ \mathbf{p}_{21} - (\mathbf{n}_1^T \mathbf{p}_{21}) \mathbf{n}_1\ _2^2; \mathbf{n}_2^T \mathbf{n}_1]$	$[d_{\min}^2; 1]$	$[d_{\max}^2; 1]$
L ₁	Pl ₂	Coinc	$[\mathbf{n}_2^T (\mathbf{p}_{21} - (\mathbf{n}_1^T \mathbf{p}_{21}) \mathbf{n}_1)]$	[0]	[0]
Pt ₁	Pl ₂	Dist (d_{\min}, d_{\max})	$[\mathbf{n}_2^T \mathbf{p}_{21}]$	d_{\min}	d_{\max}
Pt ₁	L ₂	Dist (d_{\min}, d_{\max})	$[\ \mathbf{p}_{21} - (\mathbf{n}_2^T \mathbf{p}_{21}) \mathbf{n}_2\ _2^2]$	d_{\min}^2	d_{\max}^2
Pl ₁	Pt ₂	Dist (d_{\min}, d_{\max})	$[\mathbf{n}_1^T \mathbf{p}_{21}]$	d_{\min}	d_{\max}
Fr ₁	Fr ₂	Transf ($\mathbf{T}_{2,d}^1$)	$[\text{dist}(\mathbf{T}_2^1, \mathbf{T}_{2,d}^1)]$	–	–

¹ Pl = Plane, L = Line, Pt = Point, Fr = Frame, Coinc = Coincident, Dist = Distance, || = Parallel, Transf = Transformation, Constr. = Constrained

3.5 Iterative Constraint Solvers

Let the vector (\mathbf{t}, \mathbf{r}) denote a pose in operational space with the translation \mathbf{t} and the rotation represented as axis angles $\mathbf{r} = (\mathbf{w}, \theta)$, where $\|\mathbf{w}\| = 1, 0 \leq \theta \leq \pi$. Rigid transformations are represented as $\mathbf{x} = (\mathbf{t}, \mathbf{r})$. The generic non-linear inequality constraint solving approach in (3.4) performs optimization in the space of an optimization variable v . It requires an optimization function $f(v)$ along with functions for constraints $g_i(v)$ and their bounds $\text{lb}(g_i), \text{ub}(g_i)$. Derivatives $\partial f / \partial v, \partial g_i / \partial v$ may also be provided and can be used for the optimization routine.

$$\begin{aligned} & \arg \min_{\mathbf{v}} \quad f(\mathbf{v}) \\ & \text{subject to} \quad \text{lb}(g_i) \leq g_i(\mathbf{v}) \leq \text{ub}(g_i), \quad i = 1, \dots, m. \end{aligned} \tag{3.4}$$

This optimization problem (3.4) is then solved using the non-linear optimization utility from the NLOpt [19] library with either the derivative free COBYLA [20, 21] solver, or the derivative-based SLSQP [22, 23] solver.

Depending on the target application, different constraint formulations based on this generic approach can be used. In case of the 3D GUI used for task-level programming using CAD models (Fig. 4.1), the constraints are independent of the robot. Hence, the optimization is performed in the Cartesian space using the formulation in Section 3.5.1.

In applications involving robot control, the optimization is performed in the robot joint space ($\mathbf{v} = \mathbf{q}$). This includes cases with redundant robot, where the additional DoFs in the joint space are utilized for secondary tasks or posture optimizations.

In case of prioritized tasks, the specialized solver formulation in Section 3.5.3 is used. This is the most generic formulation which is equivalent to the other formulations in some special cases. When the minimization of the joint position distance is at the second priority level and all others are at the first priority level, the prioritized solver is equivalent to the formulation 3.5.2.

3.5.1 Optimization in operational space

In this formulation, optimization is performed over transformations in Cartesian space ($\mathbf{v} = \Delta \mathbf{x} = (\mathbf{t}, \mathbf{r})$). The function to be optimized is the distance of the target pose from the current pose, since we seek the smallest transformation that satisfies all constraints. In other words, the minimization function in (3.4) is $f(\mathbf{x}) = \|\Delta \mathbf{x}\|_2^2$.

The resulting optimization problem is:

$$\begin{aligned} & \arg \min_{\Delta \mathbf{x}} \quad \|\Delta \mathbf{x}\|_2 \\ & \text{subject to} \quad \text{lb}(g_i) \leq g_i(\mathbf{x}) \leq \text{ub}(g_i), \quad i = 1, \dots, m. \end{aligned} \tag{3.5}$$

3.5.2 Optimization in configuration space

In this formulation, the optimization is performed over the target configurations of the robot ($\mathbf{v} = \mathbf{q}$). Since some of the task constraints are functions of relative transformations $g_i(\mathbf{x})$, the task derivative $\frac{\partial g_i}{\partial \mathbf{q}}$ is computed using the robot forward kinematics function FK (3.6) and central differences (3.7) with a step $\Delta \mathbf{q} = 1.0e^{-7}$

$$\mathbf{x} = \text{FK}(\mathbf{q}) \tag{3.6}$$

Input: Optimization variable \boldsymbol{v} , Constraint functions \mathbf{g}_k , their bounds $\text{lb}(\mathbf{g}_k) \leq \mathbf{g}_k(\boldsymbol{v}) \leq \text{ub}(\mathbf{g}_k)$ ($1 \leq k \leq m$) and priority levels $[1, \dots, n]$

Output: \boldsymbol{v}

```

for priority level  $1 \leq i \leq n$  do
    foreach constraint  $\mathbf{g}_k$  do
        if  $\text{Priority}(\mathbf{g}_k) > i$  then  $T_{\text{lp}} \leftarrow T_{\text{lp}} \cup \mathbf{g}_k$  ;
        if  $\text{Priority}(\mathbf{g}_k) < i$  then  $T_{\text{hp}} \leftarrow T_{\text{hp}} \cup \mathbf{g}_k$  ;
        if  $\text{Priority}(\mathbf{g}_k) = i$  then
            if  $\mathbf{g}_k$  is bounded then
                 $T_{\text{hp}} \leftarrow T_{\text{hp}} \cup \mathbf{g}_k$ 
            else
                 $T_{\text{lp}} \leftarrow T_{\text{lp}} \cup \mathbf{g}_k$ 
            end
        end
    end
    if  $T_{\text{lp}} = \emptyset$  then  $f = \|\boldsymbol{v}\|_2$  else using (3.9);
    solve for  $\boldsymbol{v}$  using (3.10)
    foreach constraint  $\mathbf{g}_k$  do
        if  $\text{Priority}(\mathbf{g}_k) = i$  &  $\mathbf{g}_k$  is unbounded then
             $\text{ub}(\mathbf{g}_k) \leftarrow \mathbf{g}_k$ 
        end
    end
end
    
```

Algorithm 2: Constraint solving for prioritized tasks

$$\frac{\partial \mathbf{x}}{\partial \mathbf{q}} = \frac{g_i(\text{FK}(\mathbf{q} + \Delta \mathbf{q})) - g_i(\text{FK}(\mathbf{q} - \Delta \mathbf{q}))}{2\Delta \mathbf{q}} \quad (3.7)$$

The function to be minimized is the distance from the current joint position $\mathbf{q}_{\text{current}}$ to the target joint position \mathbf{q} , i.e., $f(\mathbf{q}) = \|\mathbf{q} - \mathbf{q}_{\text{current}}\|_2$ in (3.4).

The resulting optimization problem is:

$$\begin{aligned} \arg \min_{\mathbf{q}} \quad & \|\mathbf{q} - \mathbf{q}_{\text{current}}\|_2 \\ \text{subject to} \quad & \text{lb}(g_i) \leq g_i \leq \text{ub}(g_i), \quad i = 1, \dots, m. \end{aligned} \quad (3.8)$$

3.5.3 Task priorities and nullspaces

In the previous sections, tasks are modeled as constraints \mathbf{g} in the optimization problem, while the optimization function f tries to find the closest robot pose in operational or configuration space that satisfies the constraints. Hence, all task constraints are modeled at the same priority level. While this approach allows finding a solution that satisfies all the constraints simultaneously, it also has some limitations. The optimization function would simply not converge in case some constraints are not satisfiable. In such cases, it is important to define a priority on the constraints so that the optimization of some unsatisfiable constraints can be sacrificed in favor of higher priority constraints.

3. CONSTRAINT SOLVERS

To achieve this, the constraint solving framework is modified. In the new formulation (3.10), the high-priority tasks (T_{hp}) are modeled as constraints g while the low-priority task (T_{lp}) are considered as costs that need not be fully satisfied, but minimized. Hence, their distance from their bounds are added to the optimization function f (3.9). Unbounded (minimization) constraints in T_{hp} are converted to bounded constraints by setting their bounds to be their function value g achieved in the previous iteration. This ensures that the optimal values achieved by these high-priority minimization constraints are not sacrificed by lower-priority constraints.

By iterating through several priority levels j , setting T_{hp} as the set of all constraints with priority $i \leq j$ and T_{lp} as the set of all constraints with priority $k > j$, this approach is extended to support multiple priority levels (Algorithm 2).

$$f_i = \begin{cases} 0 & : \text{lb}(g_i) \leq g_i \leq \text{ub}(g_i) \\ \|g_i - \text{ub}(g_i)\|_2^2 & : g_i > \text{ub}(g_i) \\ \|\text{lb}(g_i) - g_i\|_2^2 & : g_i < \text{lb}(g_i) \end{cases} \quad (3.9)$$

The resulting optimization problem is then

$$\begin{aligned} \arg \min_{\boldsymbol{v}} \quad & \sum_{g_i \in T_{\text{lp}}} f_i \\ \text{subject to} \quad & \text{lb}(g_k) \leq g_k(\boldsymbol{v}) \leq \text{ub}(g_k), \quad g_k \in T_{\text{hp}}. \end{aligned} \quad (3.10)$$

Note that this prioritized formulation of the optimization problem is computationally more expensive since it involves multiple optimization steps, one for each priority level. Hence, for a task with n priority levels, the optimization routine is called n times. The time scaling of the optimization routine depends more on the dimensionality of the state space, i.e. the number of elements in \boldsymbol{v} . This remains constant in all the n calls to the optimization routine. The number of constraints and complexity of the optimization function $f(\boldsymbol{v})$ changes with each step. Also note that consecutive optimization problems may become simpler since they start from a seed solution from the previous iteration.

3.5.4 Parameters for iterative solver

Our iterative solver formulation is based on the NLOpt library. This Section describes the parameters important for the solvers in NLOpt. One set of these parameters defines the stopping criteria for the optimization:

- `stopval` specifies the lower bound on the value of the objective function. If the objective function reaches this value, the iterative solver is deemed converged.
- `f_tol_rel` specifies the relative tolerance on the change in the value of the objective function over consecutive iterations. If this change is less than `f_tol_rel` multiplied by the absolute value of the objective function, the solver terminates.
- `f_tol_abs` specifies the absolute tolerance on the change in the value of the objective function over consecutive iterations. If this change is less than `f_tol_abs`, the solver terminates.
- `x_tol_rel` specifies the relative tolerance on the change in the value of the optimization parameters over consecutive iterations. If this change is less than `x_tol_rel` multiplied by the absolute value of the parameter for every parameter, the solver terminates.

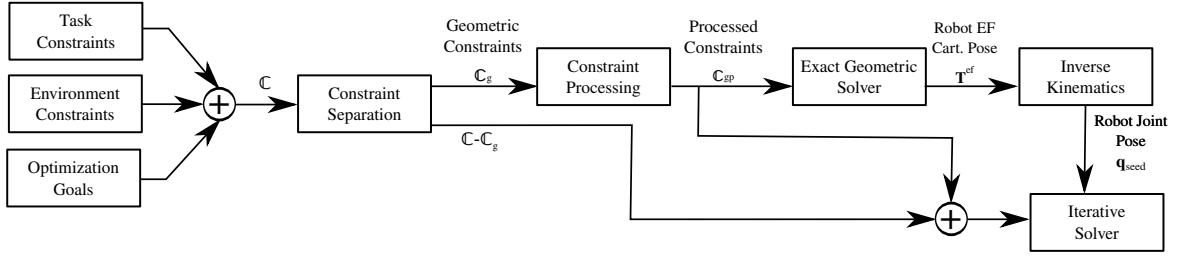


Figure 3.6: Pipeline for hybrid constraint solver, involving both exact and iterative solving steps.

- `x_tol_abs` specifies the absolute tolerance on the change in the value of the optimization parameters over consecutive iterations. If this change is less than `x_tol_abs` for every parameter, the solver terminates.
- `max_time` specifies the maximum runtime for the solver. If this time is exceeded, the optimizer terminates.
- `constraint_tol` specifies the permissible violation of the equality or inequality constraints.

3.6 Hybrid Constraint Solver

The hybrid constraint solver uses a combination of exact and iterative solvers. The overall pipeline of this approach is described in Fig. 3.6. The algorithm is formulated in Algorithms 3 and 4.

The first part of the algorithm uses the constraint simplification and combination rules from the exact solver, as described in Algorithm 3. This simplifies the geometric constraints, performs consistency checks and removes redundancies. The simplified set of constraints is then solved using the exact solver to generate a valid pose for the robot in the Cartesian space \mathbf{T}_f^{ef} , as described in Algorithm 4. Using Inverse Kinematics, the corresponding joint pose for the robot (\mathbf{q}_{seed}) is calculated. Note that the pose in Cartesian space \mathbf{T}_f^{ef} might not be reachable for the robot. In this case, we perform direct sampling on the transformation manifold that represents the task nullspace for the geometric constraints. We sample random poses in the Cartesian space on this nullspace using the parametric form of the transformation manifold, followed by an Inverse Kinematics step to check whether the sampled position is reachable for the robot. This direct sampling step is repeated until a valid (\mathbf{q}_{seed}) is obtained.

The environment constraints and optimization goals are formulated as non-linear inequality constraints suitable for an iterative solver. The simplified set of geometric constraints from the exact solver are added to this. The iterative solver is initialized with the seed position for the robot (\mathbf{q}_{seed}). The iterative solver then computes the final solution that satisfies all constraints and optimization goals.

3. CONSTRAINT SOLVERS

```

Input: Geometric Constraints  $\mathbb{C}$ 
Output: composite geometric constraints  $\mathbb{C}_g$ 
solvable  $\leftarrow$  false
foreach  $C_i \in \mathbb{C}$  do // separate constraints into R,t (Table 3.1)
| if  $\exists$  separation( $C_i$ ) then
| |  $\mathbb{C} \leftarrow (\mathbb{C} - C_i) \cup$  separation( $C_i$ )
| end
end
foreach pair  $C_i, C_j \in \mathbb{C}$  do // reduce constraints using combination rules (Table 3.2)
| if  $\exists$  combinationRule( $C_i \cup C_j$ ) then
| |  $\mathbb{C} \leftarrow (\mathbb{C} - (C_i \cup C_j)) \cup$  combinationRule( $C_i \cup C_j$ )
| end
| if  $\exists$  manifoldMap( $\mathbb{C}$ ) then // map constraint set to manifold (Table 3.1)
| | solvable  $\leftarrow$  true
| | break
| end
end
if solvable then
| if separable(manifold) then // separate into R, t constraints (Table 3.1)
| |  $\mathbb{C}_g \leftarrow$  separation( $\mathbb{C}$ )
| else // combined geometric constraint
| |  $\mathbb{C}_g \leftarrow$  projectionConstraint(manifold)
| end
end

```

Algorithm 3: Hybrid Constraint solver constraint processing rules: `separation` is a set of constraint decomposition rules defined in Table 3.1. `combinationRule` is a set of constraint combination rules, some of which have been defined in Table 3.2 and illustrated in Figs. 3.3 and 3.4. The geometric constraints \mathbb{C} are processed and solved to generate a transformation manifold that is represented as a combined constraint \mathbb{C}_g .

Input: Geometric Constraints \mathbb{C}_g , starting pose for the robot EF T_f^c

Output: seed joint position for the robot q_{seed}

```

manifold ← manifoldMap(C)
if separable(manifold) then                                // project on R, t manifolds (Table 2.4)
     $R_f^{ns(c)} \leftarrow \text{geomProject}(R_f^c) * (R_f^c)^T$ 
     $t_f^{ns(c)} \leftarrow \text{geomProject}(t_f^c) - (\text{geomProject}(R_f^c) * (R_f^c)^T * t_f^c)$ 
     $T_f^{ns(c)} \leftarrow (R_f^{ns(c)}, t_f^{ns(c)})$ 
else                                                        // project on transformation manifold
     $T_f^{ns(c)} \leftarrow \text{geomProject}(T_f^c)$ 
end
 $T^{ef} \leftarrow T_f^{ns(c)}$ 
if IK( $T^{ef}$ ) then                                        // check whether IK solution exists
     $q_{\text{seed}} \leftarrow \text{IK}(T^{ef})$ 
else
    while true do                                        // do direct sampling + IK until valid  $q_{\text{seed}}$  is found
         $T^{ef} \leftarrow \text{directSampling}(\text{manifold})$ 
        if IK( $T^{ef}$ ) then
             $q_{\text{seed}} \leftarrow \text{IK}(T^{ef})$ 
            break
        end
    end
end
    
```

Algorithm 4: Hybrid Constraint solver generating seed position in robot joint space: `manifoldMap` is the mapping of constraints to transformation manifolds, as defined (not exhaustively) in Table 3.1. `manifoldMap` is an Inverse Kinematics function that calculates robot joint positions from a given Cartesian position of the end-effector. `directSampling` is a function that generates random Cartesian positions on a given geometric manifold, using its parameteric form described in Tables 2.4 and 2.5

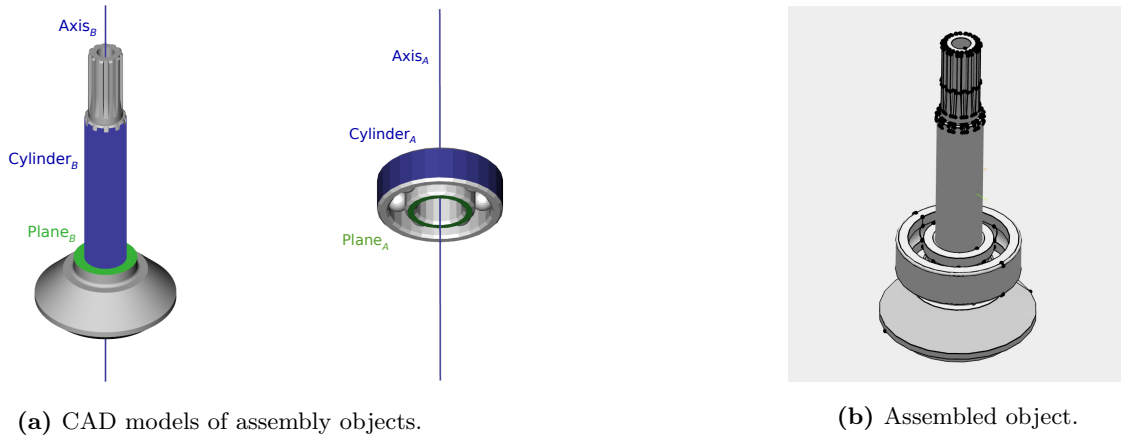


Figure 3.7: Definition of a 2-object assembly task using geometric constraints

3.7 Constraint-based Modeling Examples

In this section, we present several modeling examples using constraints. Here, we focus on the constraint-based definition of these problems. We present our software tools for modeling such constraints, solving and visualizing them. The details of their actual execution using a robotic system will be covered in the Applications part of the thesis (Parts II and III)

3.7.1 Modeling assembly tasks using geometric constraints

Modeling of object assemblies using geometric constraints is already popular in many CAD software. Such assemblies can also be modeled using our framework. Here, we present an example of assembling two objects using geometric constraints defined between the primitive shapes that comprise the object (Fig. 3.7). Fig. 3.7a shows the CAD models of the objects to be assembled. Fig. 3.7b shows the desired assembly of these objects. This assembly can be formulated using two geometric constraints: a Plane-Plane Coincident constraint between the two Planes on the objects (highlighted in green), and a Cylinder-Cylinder-Coincident constraint between the two Cylinders on the objects (highlighted in blue). Note that this assembly has one degree of freedom, i.e. rotation along the axis of the Plane or Cylinder.

Based on this example, the following Sections explain our software modules that enable definition, loading and 3D visualization of this task. We also show how our software framework can be integrate with intuitive programming interfaces that enable easy definition of these constraints through a 3D GUI.

3.7.2 Constraint modeling in JSON format

In addition to the intuitive CAD-based GUI seen in Fig. 3.9, the constraints for a task can also be defined in JSON format and loaded into our constraint solving library. Listing 3.1 shows an example of a Plane-Plane-Coincident constraint, used in the first step of the assembly constraint in Fig. 3.9a, defined using this format. The solver type also can be specified in the JSON file, as “exact”, “quaternions”, “axis-angles”, or “hybrid”.

Listing 3.1: Definition of a plan-plane coincident constraint in JSON format

```
1 {
2   "solver": "exact",
3   "constraints":
4   [
5     {
6       "geometries": "plane_plane",
7       "type": "coincident",
8       "flip": false,
9       "fixed":
10      {
11        "poseIndex": 0,
12        "point": [0, 0, 0],
13        "normal": [0, 0, -1]
14      },
15      "constrained":
16      {
17        "poseIndex": 1,
18        "point": [0, 0, 0],
19        "normal": [0, 0, 1]
20      }
21    }
22  ]
23 }
```

Additional details about the objects such as “id”, “name”, and initial pose (“transformation”) can also be specified in the JSON file, as shown in Listing 3.2. A VRML file for each object can also be specified. This 3D model is used by the GUI described in Section 3.7.3 to visualize the objects and solutions from the constraint solver. One of the objects must be specified as fixed through the “fixedPoseId” field. The constraint solver then computes the poses of constrained objects w.r.t this fixed frame. Note that this constraint definition can be extended for multiple objects, identified through their “poseIndex”, which is the “id”.

In case of iterative solvers (“quaternions”, “axis-angles”, or “hybrid”), additional parameters for the NLOpt optimizer such as the time limit (“max-time”) and tolerances for the objective function (“f-tol”, and “stopval”), the derivatives (“x-tol”) and constraints (“constraint-tol”) can also be specified in the JSON file (Listing 3.3). These parameters are described in Section 3.5.4.

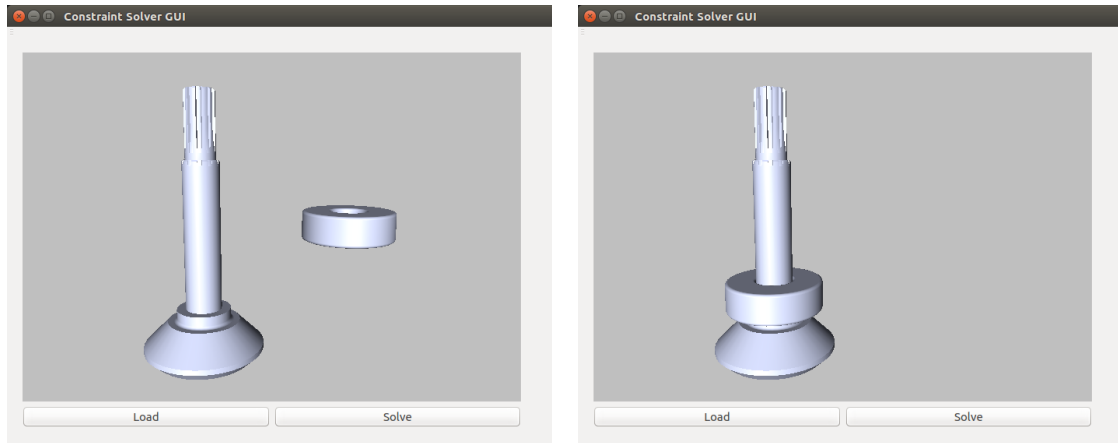
3. CONSTRAINT SOLVERS

Listing 3.2: Definition of starting poses for objects

```
1 {
2   "fixedPoseId": 1,
3   "objectPoses":
4   [
5     {
6       "id": 0,
7       "file": "mechanical-tree.wrl",
8       "transformation":
9       [
10          [1, 0, 0, 0],
11          [0, 1, 0, 0],
12          [0, 0, 1, 0],
13          [0, 0, 0, 1]
14        ]
15      },
16      {
17        "id": 1,
18        "file": "bearing.wrl",
19        "transformation":
20        [
21          [1, 0, 0, 0],
22          [0, 1, 0, 0],
23          [0, 0, 1, 0],
24          [0, 0, 0, 1]
25        ]
26      }
27    ]
28 }
```

Listing 3.3: Definition of solver parameters in JSON format

```
1 {
2   "solver-params":
3   {
4     "f-tol": 1e-6,
5     "x-tol": 1e-6,
6     "constraint-tol": 1e-6,
7     "stopval": 1e-6,
8     "max-time": 50e-3
9   }
10 }
```



(a) Load objects and constraint definitions from JSON file. (b) Solve the geometric constraints and visualize the assembled object.

Figure 3.8: GUI for loading and 3D visualization of geometric constraints between objects.

3.7.3 3D Visualization of constraint solving

We also developed a simple GUI (Fig. 3.8) for the users to load object models and corresponding JSON files defining the constraint between them. We choose the assembly example in Fig. 3.9 to demonstrate the functionality of this GUI. The loading and 3D visualization is illustrated in Fig. 3.8a. Once loaded, the “Solve” button can be pressed to solve the geometric constraints and update the 3D visualization with the final pose of the assembly, as shown in Fig. 3.8b.

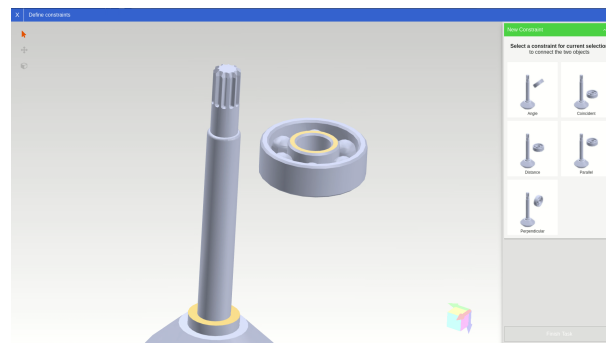
3.7.4 Modeling geometric constraints in an intuitive 3D GUI

Our constraint modeling and solving framework has been integrated into an intuitive 3D GUI [24, 25] for definition of assembly tasks using geometric constraints, as shown Fig. 3.9). In the first step (Fig. 3.9a), a Plane-Plane Coincident constraint is defined between the two Planes on the objects (highlighted in yellow). Note that once two geometries (Planes in this example) are selected, all possible and supported constraints between these two geometries are automatically computed and displayed with previews on the right side of the GUI. In the second step (Fig. 3.9a), two Cylinders are selected and a Cylinder-Cylinder-Coincident constraint is defined between them. The final pose for the assembled objects is shown in Fig. 3.9c. Note that this assembly has one degree of freedom, i.e. rotation along the axis of the Plane or Cylinder.

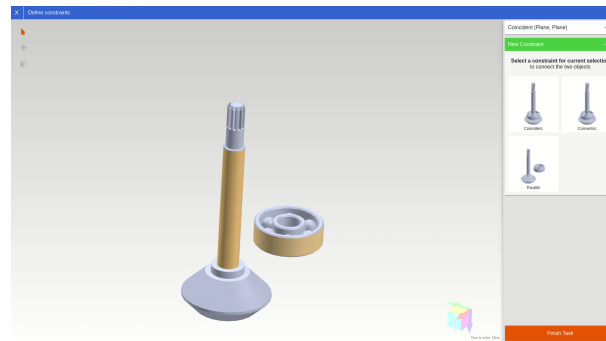
3.8 Discussion

In this Chapter, we presented three constraint solving approaches: iterative, exact and hybrid. Each solver has its own advantages and disadvantages. The choice of solver depends on the application. This Chapter concludes the basic mathematical models and algorithms for this thesis. The next part of this thesis focuses on applications of these models and algorithms in different application domains.

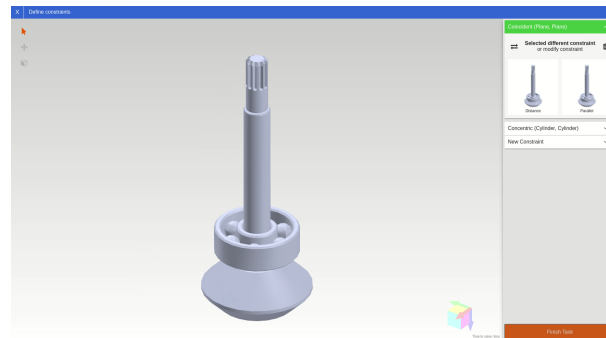
3. CONSTRAINT SOLVERS



(a) Add constraint: Plane-Plane Coincident



(b) Add constraint: Cylinder-Cylinder Concentric



(c) Final pose of the assembled objects

Figure 3.9: Constraint-based modeling example: An assembly task described using geometric constraints

The next four Chapters 4, 6, 7 and 5 each present an application domain. Chapter 4 forms the most important application of this thesis: robot motion control. All three solvers are applicable to this application domain. We present a detailed comparison and benchmarks between the solvers using typical robotics motion control tasks. Chapters 6, 7 and 5 focus on the use of our exact solver, since it is better suited to these applications and performs significantly better than the other two.

Part II

Applications in Robotics

Chapter 4

Task Level Robot Programming

This chapter presents the primary application of our constraint-based modeling techniques, i.e. task level robot programming. We demonstrate how a wide variety of robotic tasks can be expressed using constraints. We further show how this modeling technique proves beneficial to the development of intuitive programming interfaces for human-robot interaction in the front end and real-time task space robot controllers in the back-end.

4.1 Introduction

The modeling techniques used for representing robotic tasks heavily influence not only the capabilities of the robot controller, but also the intuitiveness of the robot's programming interface. The representation of robotic tasks needs to be expressive enough to capture the large diversity of robotic tasks, while being amenable for use in intuitive robot programming interfaces. In this work, we propose the use of a constraint-based modeling approach for robotic tasks, where each robot task is expressed as a non-linear constraint with inequalities, along with a priority level indicating its relative importance. In Chapter 2, we explained these models in detail. Based on these models, three different constraint-solving formulations were presented in Chapter 3. In this Chapter, we present implementations of practical robotic tasks based on these constraint models and solvers. Using the applications as benchmarks, we also compare the different constraint solving approaches.

One of the key ideas in this work is the use of an object-centric programming approach, where the robotic task is described using properties of the involved objects rather than the specific details of the manipulation process itself. Hence, goals of robotic tasks such as assembly of workpieces can be described in the form of geometric inter-relational constraints between the individual vertices, edges and surfaces present in the CAD models of the involved workpieces. Similarly, grasping tasks can be represented using geometric constraints between the robot's tool and the object to be grasped. This category of constraints, that define the task itself, needs to be explicitly specified by the user and can be achieved through various intuitive CAD-based user-interfaces.

In order to perform the specified task, the robot needs to consider its surroundings and the constraints they impose on its motion. For example, collisions with obstacles in the scene need to be avoided. Also, the robot's model and kinematic structure adds restrictions such as joint angle, velocity and acceleration

limits. Task specific goals such as manipulability maximization are also important for optimal task execution. In this work, we propose a constraint based definitions of all these aspects. Given this variety in the types of constraints and the complexity of representing them mathematically, we have observed that the modeling approach must support non-linear constraints with inequalities.

4.2 Context

The primary aim of this Chapter is to develop the concept of task-level robot programming. We motivate and demonstrate this by formulating several typical robotic applications using constraints. The concept of constraint-based robot programming can be used at different levels of robot programming. Firstly, *task or object-level programming* where a 3D GUI similar to CAD software can be used to specify geometric constraints between the involved manipulation objects. In this case, the geometric constraints can also be solved in the Cartesian space for instant feedback through this GUI. Secondly, *offline programming* of a robotic workcell, where the tasks are described as constraints between the manipulation objects, the robot and the environment. Finally, in case of applications requiring closed loop, reactive and real-time control (e.g. dynamic collision avoidance), the constraint solvers can be used as real-time controllers.

The properties and performance of the three constraint solvers proposed in this work are also evaluated in the context of these different levels.

4.3 Related Work

The roots of our work arise from the early works on operational space control by Khatib [26]. The ideas of constraint-based control start from early works such as the task-level planner Handey [27]. This was followed by works on multiple objective prioritized control and Whole Body Control Framework (WBCF) by Sentis in [28], and more recently in [29, 30].

The iTaSC framework for constraint-based control was presented in [31]. Borghesan et al. [32] presented the use of iTaSC for manipulation tasks. Decre et al. [33] extended the iTaSC framework to include inequality and non-instantaneous constraints. The eTaSL/eTC framework [34] introduced the concept of expressions graphs for defining robot tasks. The stack of tasks [35, 36] framework by Mansard et al. is another constraint-based framework that is popular for humanoid robots.

In classical literature, constraint-based methods have been most popularly used for highly redundant robots, especially humanoids. In these applications, the task space dimensions are much lower compared to the robot's joint space. Hence, task priorities are often handled using projections and local approximations. Dietrich et al. [37] present an overview of such approaches. Multi-task optimization in the context of humanoid robots was presented in [38] and [39]. The Task Space Inverse Dynamics (TSID) [40] approach by del Prete et al. is an efficient approach for task-space control using local linearizations. Some other approaches based on these concepts are described in [28, 39, 41, 42, 43]. Such approaches have also been extended to humanoid robots on mobile bases [44]. This concept is not ideal for industrial manipulators since there are few robot joints (typically 6-7) compared to the task nullspace for effective utilization of multiple levels of nullspace projections. In such applications, approaches involving quadratic (or non-linear in general) solvers are more popular. Efficient solvers for such prioritized optimal control tasks were presented in [45] and [30].

In terms of the types of constraints and solvers used, most robotics frameworks optimize in the linear least squares sense, e.g., operational space control [26], iTaSC [31], and more recently in Task-Space Inverse Dynamics [40]. However, most tasks are naturally expressed with inequality constraints, e.g., joint limits [46], collision avoidance [47, 48], and singularity avoidance [16]. A number of frameworks can handle inequality constraints [33, 49], but do not allow prioritized tasks. Escande et al. [30] presented a specialized optimal solver that can handle *linear* inequality constraints with multiple levels of priorities. Many robotic constraints are non-linear by definition (e.g., manipulability [16]) and the constraint solving framework needs to support this explicitly.

Applications based on a representation of coordinate frames and geometric relations between them were presented in [50, 51]. Rodriguez et al. [11] defined tasks using geometric relations between geometric entities (e.g., points, lines, surfaces) that comprise manipulation objects. The concept of uncertainties in geometric representation of features and relations was presented in early works such as Durrant-Whyte [52].

Kresse et al [42, 43] presented a constraint-based framework for specifying robotic manipulation tasks. In addition to geometric constraints and kinematic control, the framework also supports force control. This framework uses an interaction matrix generated from the task description to compute the task space velocities, that are combined using prioritized nullspace projections.

Recently, [53] presented formal language definitions for constraint-based tasks descriptions, aimed towards use with symbolic planners and reasoning engines. Perzylo et al. [2] showed how semantic descriptions of geometries and constraints can be used for robot task definitions. This was further extended to semantic descriptions of complete robotic tasks including the environment/workcell and automatic reasoning and planning [4].

In terms of the programming approach for robots, we also derive our work from concepts such as motion primitives, robot skill definitions and trajectory optimization. The idea of motion primitives has been popular since early works in [54, 55] till recent ones in [56], [57, 58] and [59]. Robot skills, especially in the context of semantic descriptions have been presented in several recent works [53, 59, 60, 61]. SkiROS [62] is a framework based on the Robot Operating System (ROS) that provides a skill-based programming interface for robots. The use of constraint-based task definitions in the context of intuitive interfaces for human-robot interaction were studied in [4, 24, 25, 63]. Modern trajectory optimization frameworks such as CHOMP [64] can keep a given distance to obstacles, and the more recent Trajopt [13] allows operational constraints to be defined.

4.3.1 Contributions

The major qualitative contributions of our approach w.r.t to the state-of-art are summarized in Tables 4.1 and 4.2. Benchmarks and quantitative comparisons are presented in Section 4.7.

Table 4.1 presents a qualitative comparison of some popular constraint-based control approaches that are relevant to this work. The runtimes are roughly classified as $--$ being less than 4 ms, $-$ being between 4 ms and 1 ms, $+$ being between 1 ms and 0.1 ms, and $++$ being less than 0.1 ms. It is clear that the exact approaches are generally much faster than the iterative solvers. Solvers based on local linearizations such as TSID and WBCF are also very efficient, but are restricted in their scope of supported geometric constraints.

Table 4.1: Overall comparison of control frameworks

Framework	Efficient	Exact	Geom.Constraints	ForceControl	Inequality	Prioritized	UnderActuated	Output
TSID [40]	+			✓		✓		τ
WBCF [28]	+			✓		✓	✓	τ
Lenz et al. [48]	-				✓	✓		\dot{q}
iTaSC [31, 33]	N.A.			✓	✓	✓	✓	\dot{q}
SoT [35, 36]	N.A.			✓	✓	✓	✓	\dot{q}
Somani et al. [65]	--		✓					q or x
Rodriguez et al. [11]	++	✓	✓	✓				q
Exact [6]	++	✓	✓		✓			x
Iterative [7]	+		✓		✓	✓	✓	q or x
Hybrid	+	✓	✓		✓	✓	✓	q

54

Table 4.2: Comparison of geometric constraint solvers

Framework	Non-separable R,t	Inequalities	CyclicDependencies	Prioritized	Use-Cases (Table 4.5)	OutputType
Rodriguez et al. [11]					A,C	x
Exact [6]	✓	✓			A,B,C,D,E,F	x
Hybrid	✓	✓	✓	✓	A,B,C,D,E,F	q or x

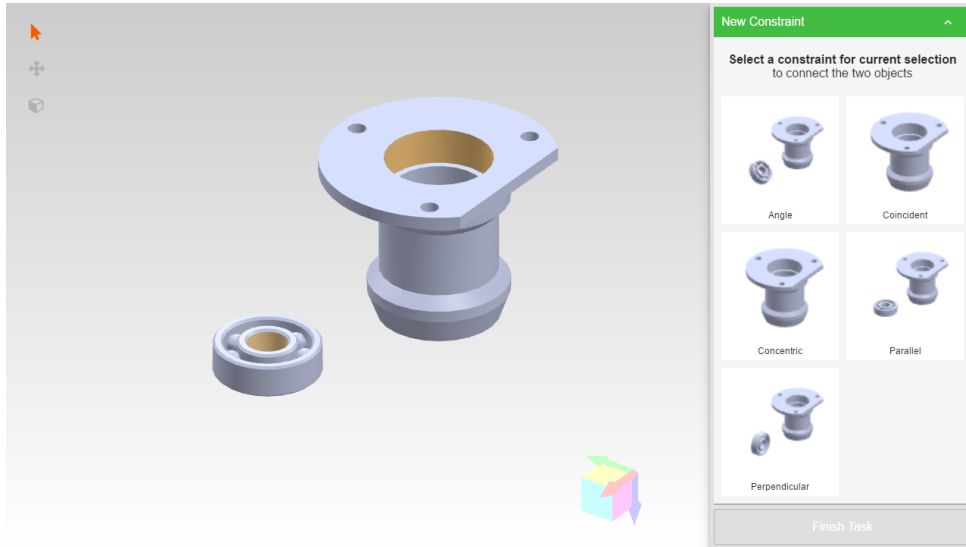


Figure 4.1: Intuitive user-interface for definition of geometric constraints between individual shape elements of CAD models. After selecting a pair of valid surfaces, the user is presented with a preview of all valid constraints between the selected surfaces.

Although many of the mentioned frameworks support some geometric constraints, the complexity and variety of constraints that they support and mention in their work are rather limited. The exact solver by Rodriguez et al. [11] is one approach where geometric constraints are the core focus and can be compared to our work. Since our work also focuses largely on geometric constraints, a dedicated comparison of geometric solvers is presented in Table 4.2. It lists our improvements over existing frameworks and the core qualitative contributions of our exact geometric solver. Firstly, we support mixed transformation manifolds, i.e. cases where the rotation and transformation components of the transformation manifolds are not independent. Secondly, we support geometric constraints with inequalities.

We claim that our hybrid solving approach based on a combination of exact and iterative solvers can provide the best features from both types of approaches. While our exact solver already provides these advantages beyond the state-of-art, combination with an iterative solver provides it with two additional features: multiple priority levels for constraints and cyclic dependencies between constraints. Also, the hybrid solver enables inclusion of domain-specific constraints that may not be geometric.

4.4 3D GUI for Task-level Robot Programming

Our concept of constraint-based tasks specification is used to develop intuitive interfaces for human-robot interaction and robot instruction [24, 66]. This includes a web-based 3D GUI, where CAD models can be imported and visualized. Geometric inter-relational constraints between these geometric entities (e.g., workpieces, tools, robot) can be defined by selecting primitive shapes (i.e., plane, edge, point) from the object models and choosing the desired constraint from a filtered list (Fig. 4.1). In the background, the exact solver presented in Section 3.4 computes the relative transformation between the models based on the selected constraints and provides immediate visual feedback. The runtime of the constraint solver is also an important factor to ensure a smooth and reactive user experience. With the exact solver in

4. TASK LEVEL ROBOT PROGRAMMING

the background, all possible constraint types for a pair of selected geometric primitives can be quickly calculated and visualized for the user without any discernible lag(Fig. 4.1).

Using this GUI, a user can intuitively describe robotic tasks in the form of constraints. Once the task constraints have been specified, the workcell and robot to execute the constraints can be selected. At this stage, constraints arising from the robot model and the workcell/environment are added to the task specification.

To execute this task on the robotic platform, either the iterative or hybrid solver can be used. Next, Section 4.5 presents several examples of constraint-based robotic tasks. A comparison of the execution of these examples using different constraint solvers is discussed in Section 4.7.

4.5 Robot Skills

Robot skills are implementations of behaviors or capabilities of the robot. In essence, they control motion of the robot in operational/configuration space in a way that this capability or behavior is realized. In our formulation, they are expressed as constraints on the robot’s motion in the operational or configuration spaces.

We present several examples of robot skills, each highlighting a specific feature of our constraint-based approach. We highlight the distinction between task constraints such as geometric constraints, environment constraints such as collision avoidance, and task specific optimizations such as the manipulability measure.

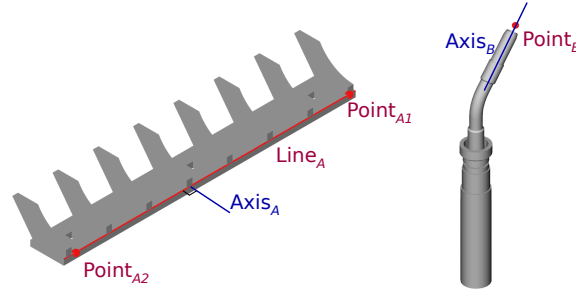
We present several details for each example. Firstly, the overall task objective is described. This is followed by the constraint-based definition of this task. Finally, an analysis of the nullspace of this task is presented. Each example contains pictures showing the involved geometrical entities, a table describing the constraints, nullspace definition and DoFs, and images of sample poses showing the solution space. We also present videos^{1 2} and pictures of implementation of these skills on robotic manipulators, both in simulation and real hardware.

4.5.1 Welding with obstacle avoidance

Welding is an intrinsically underspecified task, where the position of the tip of the welding tool is constrained with respect to the object to be welded. In case of *point welding* (Fig. 4.3a), the tip must coincide with the specified location of the spot weld. In case of *seam welding*(Fig. 4.3b), the tip of the welding tool must move along the weld seam (e.g., a line on the object) at a specified velocity between a starting and ending point. The orientation of the welding tool is free and defines the nullspace of the task. In practical situations, the orientation is not entirely free but needs to be restricted to ensure a good weld quality. Through the use of inequality constraints, these limits on the task nullspace can be defined easily.

¹<https://youtu.be/baet9IkTK04>

²<https://youtu.be/qRJ1JmNoFEw>



Seam Welding	Task Nullspace	DoF
$\text{Coincident}(\text{Point}_B, \text{Line}_A)$ $0 \leq \text{Angle}(\text{Axis}_B, \text{Axis}_A) \leq \theta_{max}$ $\text{Coincident}(\text{Point}_B, \text{Point}_{A1})$ (start point) $\text{Coincident}(\text{Point}_B, \text{Point}_{A2})$ (end point)	Rotation: $\text{OneAngleManifold}(\text{Axis}_A, \theta_{min}, \theta_{max})$ Translation: Line_A	Pose: 6 Nullspace: 2
Point Welding	Task Nullspace	DoF
$\text{Coincident}(\text{Point}_B, \text{Point}_{A1})$	Rotation: \mathbb{R}^3 Translation: Point_{A1}	Pose: 6 Nullspace: 3

Figure 4.2: Constraint for seam welding.

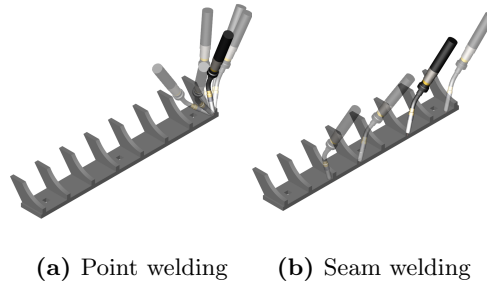


Figure 4.3: Sample poses and task nullspaces for (a) point welding and (b) seam welding.

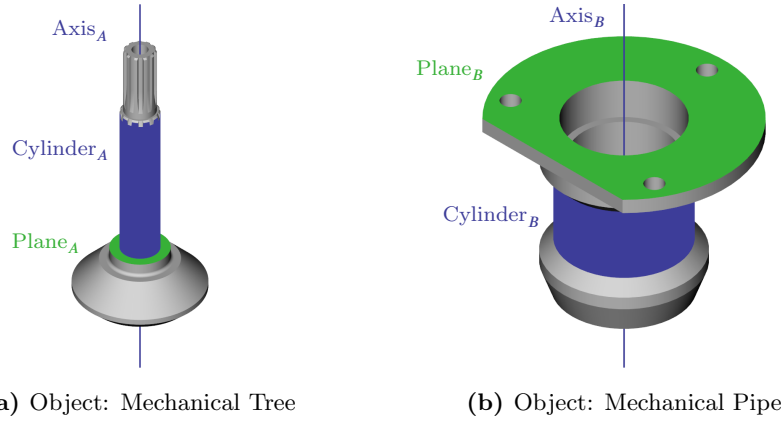
4.5.1.1 Constraint-based formulation

The constraint-based definition of this task is shown in Fig. 4.2. When this task is executed on a robotic workcell, additional constraints such as collision avoidance (Section 2.5.1) with other objects present in the workcell are added. The collision avoidance task is assigned a priority higher than the desired velocity task.

4.5.1.2 Task Nullspace

Fig. 4.3b shows some sample target poses for the welding tool and also illustrates the nullspace of its geometric constraints. The task nullspace can be utilized to satisfy a secondary objective (e.g., collision avoidance) or simply manually tuned by interactive jogging. Fig. 4.23a and Fig. 4.23b show the execution of this task in a robotic workcell in simulation. Real execution of this task on a Comau SmartSix-614 is

4. TASK LEVEL ROBOT PROGRAMMING



Assembly	Task Nullspace	DoF
Concentric($Cylinder_A$, $Cylinder_B$) Coincident($Plane_A$, $Plane_B$)	Rotation: OneParallelManifold($Axis_A$) Translation: –	Pose: 6 Nullspace: 1

Figure 4.4: Geometric constraints for the assembly of two workpieces

shown in Fig. 4.23e and Fig. 4.23f. The pose in the middle illustrates how the orientation of the tool is changed to avoid the obstacle.

4.5.2 Assembly of two workpieces

In this example (Fig. 4.4), two workpieces from a gearbox need to be assembled together. This task consists of 3 steps: (1) picking object B , (2) moving to an approach position near object A , and (3) assembling the objects together. Fig. 4.4 describes step (3), i.e., the final assembly pose for the two objects. Steps (1) and (2), i.e., grasping and moving to an approach pose, can also be defined using geometric constraints between the gripper and the objects A and B .

4.5.2.1 Constraint-based formulation

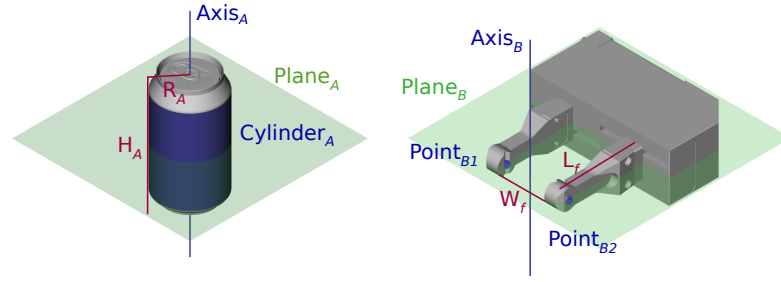
The parameters for each of these steps can be specified using geometric constraints between the assembly objects (A and B , Fig. 4.4) and the parallel gripper in Fig. 4.5 (B).

4.5.2.2 Task Nullspace

The relative position of the objects A and B in the assembly is underspecified, with one degree-of-freedom as the rotation along the $Axis_A$ of object A . Fig. 4.4 shows the definition of this geometric nullspace. Fig. 4.23d and Fig. 4.23h show the execution of this assembly task in a robotic workcell in simulation and using a Comau SmartSix-614 respectively.

4.5.3 Cup grasping with obstacle avoidance

This task involves grasping a cylindrical object (a cup). This task is demonstrated on two robots: a 6 DOF Comau RACER 7-1.4 (Fig. 4.24a), and a 7 DOF KUKA Light Weight Robot (LWR)(Fig. 4.24b).



Grasping (All Priority 1)

$$R_A \leq \text{ParallelDistance}(\text{Axis}_A, \text{Axis}_B) \leq L_f - R_A$$

$$-H_A/2 \leq \text{Distance}(\text{Plane}_A, \text{Plane}_B) \leq H_A/2$$

$$R_A \leq \text{Distance}(\text{Axis}_A, \text{Point}_{B1}) \leq W_f - R_A$$

$$R_A \leq \text{Distance}(\text{Axis}_A, \text{Point}_{B2}) \leq W_f - R_A$$

Environment/Robot Constraints

Avoid collisions (Priority 1)

Minimize Cartesian delta of end-effector (Priority 2)

Figure 4.5: A cup grasping task expressed using geometric constraints with inequalities. The rotation and translation components in this example are not independent.

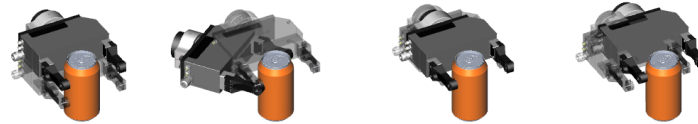
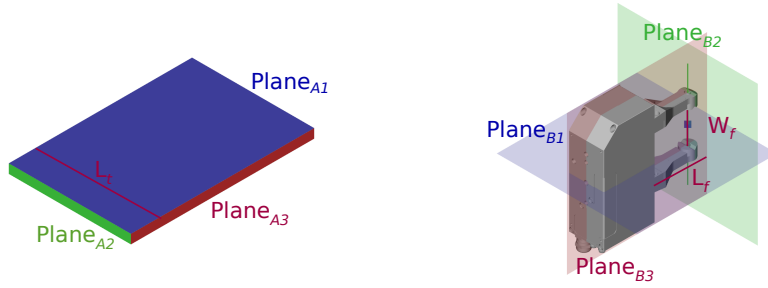


Figure 4.6: Sample poses and task nullspaces for the cup grasping task.

4.5.3.1 Constraint-based formulation

The constraint-based formulation of this task is shown in Fig. 4.5. The use of inequality constraints allows more accurate modeling of the task and better use of robot's capabilities, such as the opening width of the gripper, length of the gripper finger and the length of the cup. During execution, additional constraints from the environment such as collision avoidance are added. To ensure smooth motion, the Cartesian distance between the end-effector pose in successive motions is minimized as a low priority task. The robot utilizes the nullspace of the task to avoid obstacles. In addition to the rotation along the cylinder's axis, the min-max constraint formulation provides the robot additional degrees of freedom along the length of the cylinder (H_A) and along the length of the gripper fingers (L_f).

4. TASK LEVEL ROBOT PROGRAMMING



Tray Grasping	Task Nullspace	DoF
$-W_f/2 \leq \text{Distance}(\text{Plane}_{A1}, \text{Plane}_{B1}) \leq W_f/2$ $0 \leq \text{Distance}(\text{Plane}_{A2}, \text{Plane}_{B2}) \leq L_f$ $0 \leq \text{Distance}(\text{Plane}_{A3}, \text{Plane}_{B3}) \leq L_t$	Rotation: – Translation: $\text{Box}(L_f, L_t, W_f)$	Pose: 6 Nullspace: 3

Figure 4.7: Manipulation of a tray: The robot task is to carry a tray that contains objects. To prevent the objects from falling, the tray must be kept upright. Allowed tolerances in rotation can be encoded as min-max values of the orientations along the respective axes.

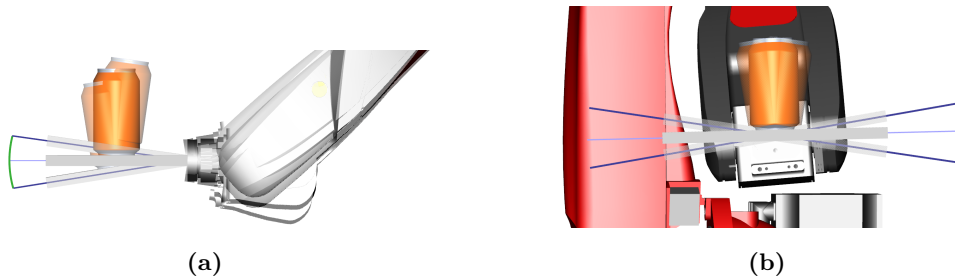


Figure 4.8: (a) and (b) The tray is grasped by a single robot arm.

4.5.3.2 Task Nullspace

The geometric nullspace of this task is defined in Fig. 2.21. This task is an example where the rotation and translation elements in the task nullspace are dependent.

4.5.4 Tray grasping with obstacle avoidance

This application involves manipulating a tray containing a can with liquid (Fig. 4.7).

4.5.4.1 Constraint-based formulation

Fig. 4.7 shows the constraint-based formulation of this task. `ParallelDistance` constraints between three sets of non-parallel Planes would, in the absence of inequalities, result in a fully specified robot pose. Hence, inequality constraints are necessary to model the nullspace of this task. The rotation and translation components of the constraints are separable. Two `ParallelDistance` constraints between sets of non-parallel Planes generate an infinite `Parallelepiped`, whose infinite axis is the line of intersection of the two Planes.

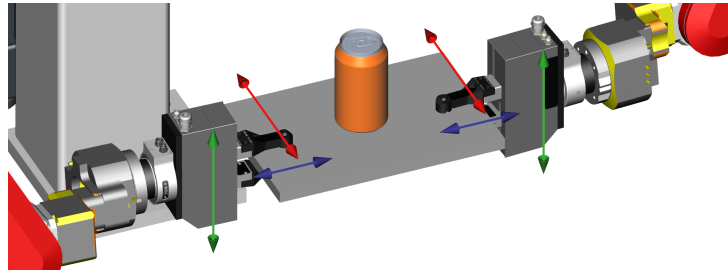


Figure 4.9: The tray is grasped by a dual-arm robot. A set of min-max constraints can be used to express the inherent flexibility of the grasping task.

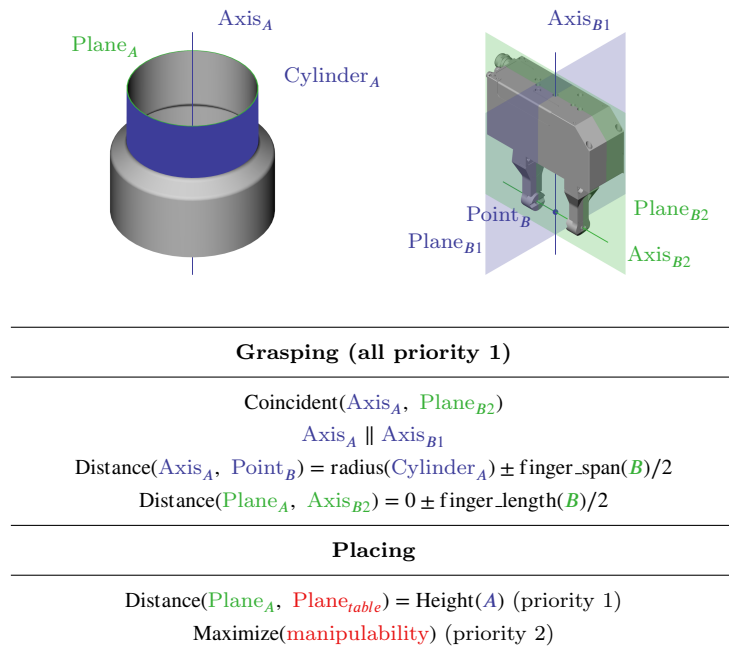


Figure 4.10: Constraints for grasping a cylindrical object at its rim and placing it on the table

Three `ParallelDistance` constraints between sets of non-parallel `Planes` generate a finite `Parallelepiped`. The rotation in this case is fully defined.

4.5.4.2 Task Nullspace

The task nullspaces are specified in Fig. 4.7. The robot can utilize this nullspace to achieve secondary objectives such as avoiding obstacles (Fig. 4.24c). This task is performed using two different robotic platforms. With a 6 DOF Comau RACER 7-1.4, the tray was grasped using the parallel gripper on one side (Fig. 4.8b). Using a Comau dual arm, the tray was grasped on two sides (Fig. 4.9).

4.5.5 Pick and place with manipulability optimization

This application consists of two tasks: grasping of a cylindrical object at its rim, and placement of the grasped object on another position on the table.

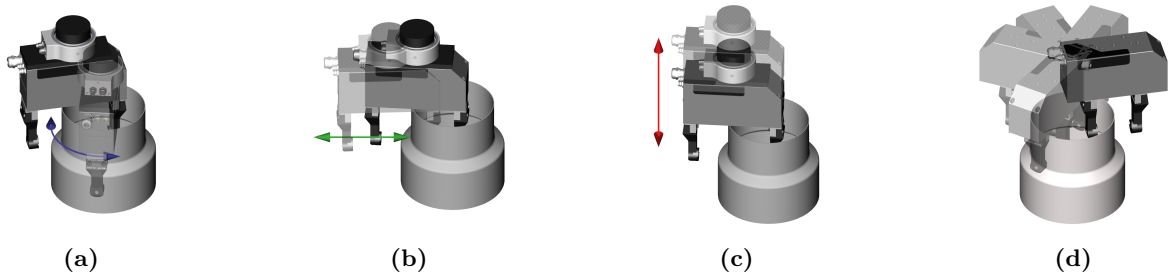


Figure 4.11: Underspecified robot tasks: the robot pose is not completely specified and can be moved in the nullspace highlighted by the dotted axes. The grasping point can be located anywhere along the rim of the cylinder (a) (blue). Also, depending on the gripper span (b) and finger length (c), it can be moved along the green and red axes respectively.

4.5.5.1 Constraint-based formulation

The grasping pose in the first step of this application is underspecified and can be further refined by jogging the robot in the nullspace of the task constraints (Fig. 4.11). In Fig. 4.24d, the first step of this application is shown with transparent objects. In the second step, the object is moved to a position on the table that maximizes its manipulability. The task constraints for the placement step state that the object should be placed in an upright orientation on the table. This is combined with lower-priority posture optimization (manipulability maximization).

4.5.5.2 Task Nullspace

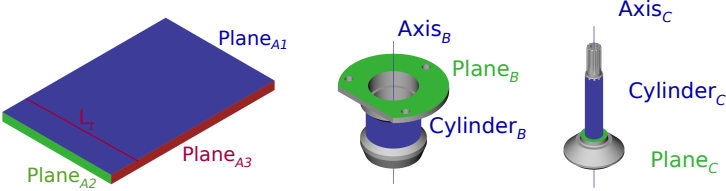
Some exemplary grasping poses and the nullspace for the task constraints are illustrated in Fig. 4.11. Two different locations of the cylindrical object on the table are used to illustrate how the underspecified grasping pose is further optimized in its nullspace to minimize the joint distance from the same starting pose (Section 3.5.2). The manipulability values of different robot poses on the table are overlaid (red indicates low, and green high) in Fig. 4.24d.

4.5.6 Manipulating a tray carrying an assembly

This task presents an example with multiple objects. The object assembly described in Section 4.5.2 is placed on a tray. The assembly itself has one rotational DoF along its symmetrical axis in its nullspace. Additionally, the object can translate in two directions along the tray and rotate around its normal direction. The tray in turn is manipulated by a robotic arm and can translate in all 3 axes as well as rotate along the normal direction of tray plane. The other two rotational axes are restricted to prevent the object from falling off the tray.

4.5.6.1 Constraint-based formulation

The gearbox assembly is defined using two constraints, as shown in Section 4.5.2. The task of placing the gearbox on the tray is defined using a PlanePlaneDistance constraint between the top surface of the tray and a plane on the gearbox. In order to restrict rotations along the X and Z axes, a PlanePlaneParallel constraints is defined between the plane on the end-effector whose normal points in the Y direction, and



Gearbox Assembly	Task Nullspace	DoF
Concentric(Cylinder _C , Cylinder _B) Coincident(Plane _C , Plane _B)	Rotation: OneParallelManifold(Axis _C) Translation: –	Pose: 6 Nullspace: 1
Object on tray Distance(Plane _{A1} , Plane _C) = 0.05m	Rotation: OneParallelManifold(Axis _C), OneParallelManifold(Normal _{A1}) Translation: Plane _{A1}	Pose: 12 Nullspace: 4
Tray on robot EF Parallel(Plane _γ , Plane _{A1})	Rotation: OneParallelManifold(Axis _C), OneParallelManifold(Normal _{A1}), OneParallelManifold(Normal _γ) Translation: Plane _{A1} , ℝ ³	Pose: 18 Nullspace: 8

Figure 4.12: Multi-object task: manipulating a tray that carries an assembled part (Section 4.5.2)

the top surface of the tray. Since there are three consecutive sets of constraints, the solution is obtained by solving the problem in three steps.

4.5.6.2 Task nullspace

The first task (assembly) has 1 DoF along the rotation axis. The second task has 3 DoFs, two in translation and 1 in rotation. The third task has 4 DoFs: 3 in translation and 1 in rotation. In total, this system has 4 constraints, 18 DoFs in the combined pose dimension and 8 DoFs in the task nullspace, as shown in Fig. 4.12.

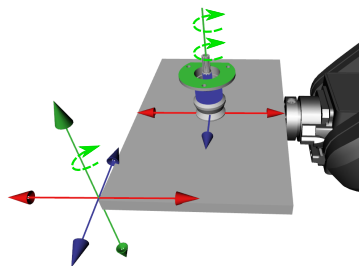


Figure 4.13: Nullspace of a multi-object task: manipulating a tray that carries an assembled part

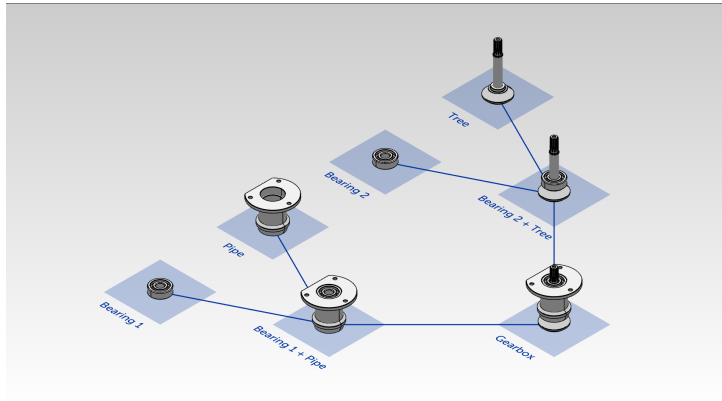


Figure 4.14: Assembly steps for gearbox assembly task

4.6 Applications

Our intuitive programming approach was also used for two robotic application domains as a part of the European FP7 project SMERobotics¹. The first application involved the assembly of a gearbox (Section 4.6.1). The second application was in the domain of woodworking [67] and involved the construction of the wall of a wooden house (Section 4.6.2).

4.6.1 Gearbox assembly

This task is an example from the assembly domain, where 4 workpieces are assembled to form part of a gearbox. Fig. 4.14 describes the different steps in this task, and ordering constraints on the steps. It is a very precise assembly operation with tolerances of approx. 1 mm. Each of the 3 steps in this assembly are programmed using our intuitive 3D CAD-based programming interface (Section 4.4). This task was executed on three robotic platforms: a Comau Racer-714, a Comau SmartSix-614, and a Comau dual arm.

4.6.1.1 Constraint-based formulation

The constraint-based formulation of each step of this task is described in Fig. 4.15. Images showing sample poses for each step are also provided. Each assembly step consists of 2 constraints. There is an ordering restriction on the steps: steps 1 and 2 have to be performed before step 3. Note that the overall task is a multi-object assembly. The 4 objects involved in the assembly form a chain of constraints. Hence, the total dimensionality of the relative poses is $3 \times 6=18$.

4.6.1.2 Task Nullspace

For the first two steps, the relative pose is defined between two objects. Hence, the relative pose is 6 dimensional. In both these steps, the nullspace is a rotation along the axis and has 1 DoF. The third step involves the assembly of two such assemblies, having pose dimensionality $6 + 6 + 6 = 18$. The nullspace DoF for this steps is $1 + 1 + 1 = 3$.

¹<http://www.smerobotics.org>

Assembly step	Task Nullspace	DoF
Assembly step 1 Concentric($Cylinder_B$, $Cylinder_A$) Coincident($Plane_B$, $Plane_A$)	Rotation: OneParallelManifold($Axis_B$) Translation: –	Pose: 6 Nullspace: 1
Assembly step 2 Concentric($Cylinder_C$, $Cylinder_A$) Coincident($Plane_C$, $Plane_A$)	Rotation: OneParallelManifold($Axis_C$) Translation: –	Pose: 6 Nullspace: 1
Assembly step 3 Concentric($Cylinder_B$, $Cylinder_C$) Distance($Plane_B$, $Plane_C$) = $0.1m$	Rotation: OneParallelManifold($Axis_B$), OneParallelManifold($Axis_B$), OneParallelManifold($Axis_C$) Translation: –	Pose: 18 Nullspace: 3

Figure 4.15: Constraints for a 4-object (2xA, 1xB, 1xC), 3-step assembly.

4.6.2 Woodworking

This application is based on an existing robot cell at a woodworking industry that manufactures wooden houses. One of the operations in their process flow is the creating wooden walls from wooden panels and frames. For this operation, the panels have to be first placed on a wooden frame. The the panels are attached to the frame by nailing along lines on the frame. Finally, the excessive segments of the panels are removed by sawing along the periphery of the frame. Fig. 4.16 shows an example of the work-flow involving these pick-and-place, nailing and sawing operations.

The physical robot is a six degrees of freedom gantry system, Fig. 4.21. The area of the worktable is approximately $30\text{ m} \times 4\text{ m}$. Stored along the worktable are palettes of panels. The robot also has access to a tool store featuring several tools, such as saw, mill, nail gun and combination tools of these.

This application also shows that constraint-based applications are not only relevant for assembly tasks but are also effective in describing nailing and sawing tasks.

4.6.2.1 Constraint-based formulation

The constraint-based formulation of the different steps in this example are shown in Figs. 4.17, 4.18, and 4.19. There are two assembly steps, one nailing task with 4 nailing lines and one sawing task with 4 sawing lines in this example. The ordering constraints for this task are illustrated in Fig. 4.16.

One step of the assembly is shown in Fig. 4.17. The nailing task is formulated as a PointLineCoincident constraint, where the tip of the nailing gun lies along a defined line on the frame. To ensure that the nails

4. TASK LEVEL ROBOT PROGRAMMING

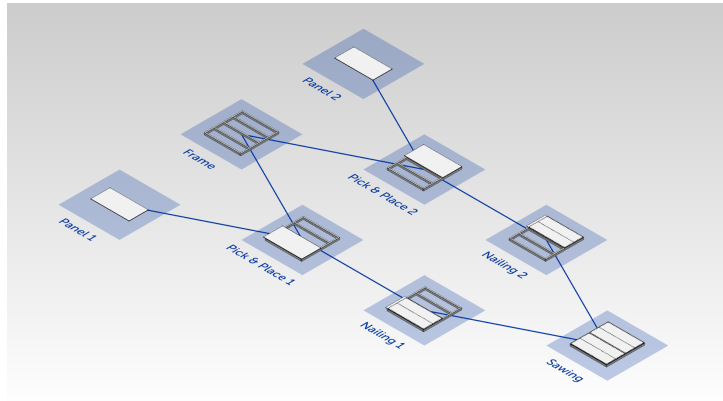
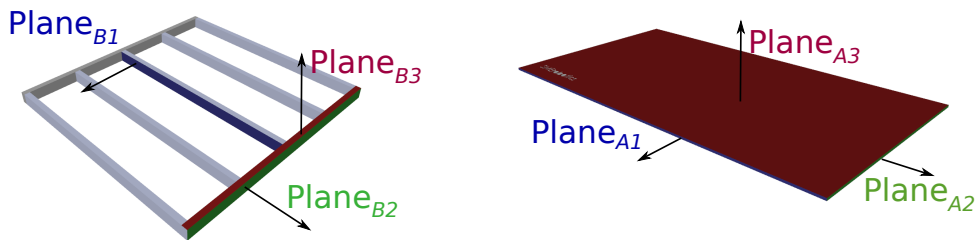


Figure 4.16: Steps and dependencies of woodworking task



Woodworking assembly	Task Nullspace
Coincident(Plane_{B1} , Plane_{A1})	Rotation: –
Coincident(Plane_{B2} , Plane_{A2})	Translation: –
Coincident(Plane_{B3} , Plane_{A3})	

Figure 4.17: Constraints for assembling a panel on a wooden frame.

are shot perpendicular to the surface of the frame, a `PlanePlaneParallel` constraint between the frame surface and a flat surface on the tool is added. Note that the nailing lines lie roughly in the middle of the wooden planks that form the frame. The constraint-based description of this task is shown in Fig. 4.18. The sawing task is formulated as a `LineLineCoincident` constraint, where the cutting direction (a line) of the sawing blade coincides with the sawing line. To ensure that the sawing blade is perpendicular to the surface of the frame, a `PlanePlaneParallel` constraint is added. Note that the sawing lines lie along the periphery of the frame. The constraint-based description of this task is shown in Fig. 4.19.

4.6.2.2 Task Nullspace

Each assembly step involves three `PlanePlaneCoincident` constraints. This results in a fully specified pose for the panels, and an empty nullspace in both rotation and translation. The nailing and sawing steps have one DoF, along the respective lines. In practice, this is not a DoF since the tool has to move along the line segment with a constant velocity.

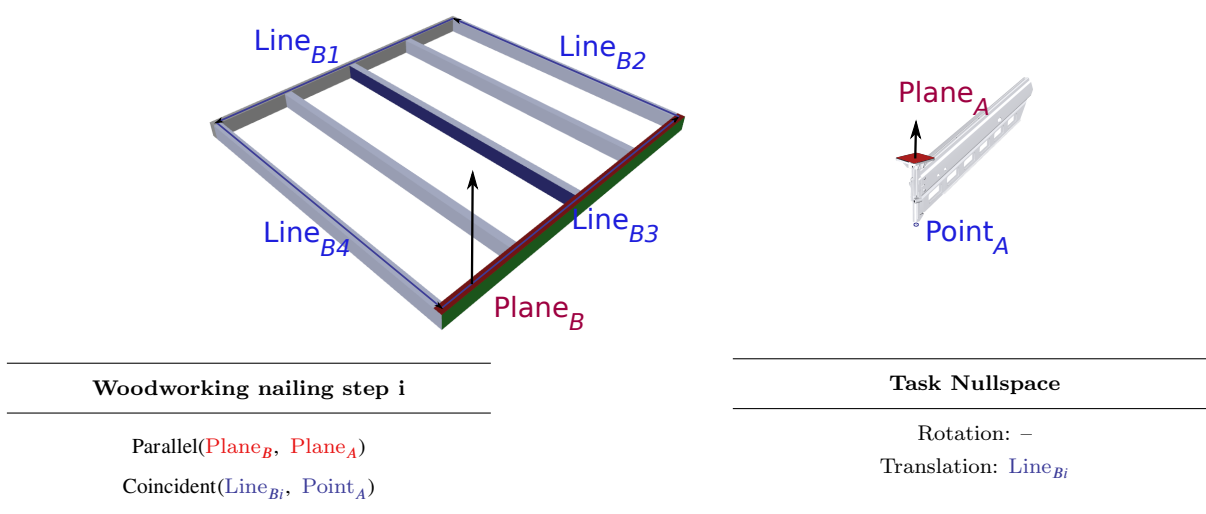


Figure 4.18: Constraints for nailing a panel to a wooden frame.

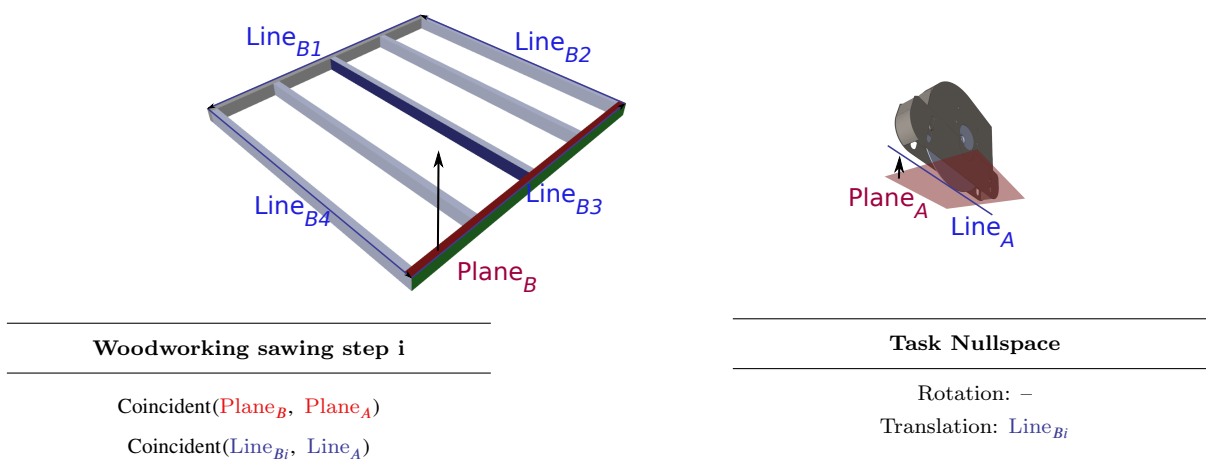


Figure 4.19: Constraints for sawing the extra edges of a panel nailed to a wooden frame.

4.7 Evaluation

We evaluated our approach on a selection of classic industrial robotic scenarios in simulation and in real-world setups (industrial robots *Comau Racer7-1.4*, *Comau Smart5 Six 6-1.4*, *Comau Dual Arm*, and *Güdel Gantry robot*). Fig. 4.23 presents some images from simulated and real experiments for some of the robot skills presented in Section 4.5. These experiments involved only the task constraints. Fig. 4.24 presents some results from simulated execution of these tasks in the presence of environment constraints such as collision detection and task optimizations such as manipulability optimization. This set of experiments also shows that our approach can be applied to redundant robots such as the 7 DoF *Kuka LWR*, and to multiple end-effectors such as the 13 DoF *Comau Dual Arm*.

4. TASK LEVEL ROBOT PROGRAMMING



(a) Execution of the assembly task on a dual arm robot.



(b) Execution of the assembly task using a 6 DoF Comau Racer 7-14 robot.

Figure 4.20: Execution of assembly task on different robotic platforms.

While the constraint modeling and solving parts of our implementation are new, inverse kinematics, trajectory generation and low-level robot control use the Robotics Library¹ by Rickert [68].

4.7.1 Intuitive robot programming

One of the claimed contributions of this work is the development of a task-based robot programming interface that is intuitive for the users. While a detailed analysis of human-robot interfaces that enable this intuitive interaction is beyond the scope of this work, we focus on the importance of constraint-based task definitions in this quest. We present excerpts of some user-studies and evaluations on real robotic applications to show that intuitive interfaces utilizing the concepts of geometric constraints for robot task definition perform better than conventional interfaces.

The presented quantitative evaluations are for the whole intuitive programming system [4]. Our constraint-based task definition framework is only one part of this system. The speed-ups achieved in these evaluations are a result of many other factors such as semantic descriptions, efficient computer vision modules, etc. It is difficult to isolate the contribution of constraint-based descriptions in these evaluations. Hence, these evaluations should be considered mainly as qualitative proof that our approach helps improve robot programming intuitiveness and time.

For these evaluations, the Exact solver for geometric constraints (Section 3.4) was used as the back-end solver for the 3D CAD mating interface (Section 4.4) to generate target Cartesian positions for the robot. The robot joint positions were calculated using inverse kinematics from the Robotics Library [68, 69].

4.7.1.1 Precise assembly of multiple parts

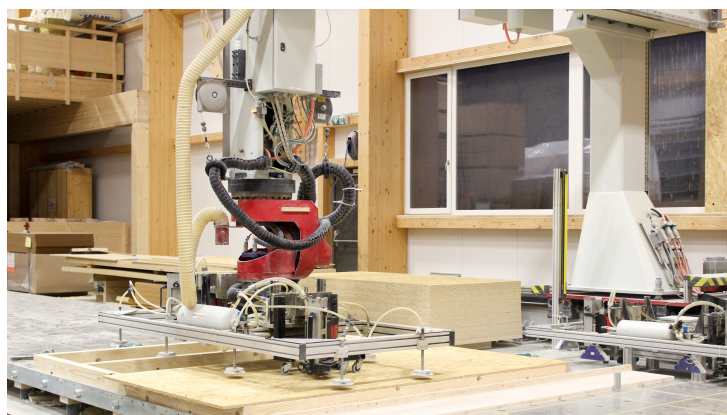
The gearbox assembly example from Section 4.6.1 is used in this evaluation. The task is executed on two different robotic platforms: a 6 DoF *Comau Racer 7-14*, and a *Comau Dual Arm* (Fig. 4.20).

For this setup, the programming of the assembly tasks is done via an intuitive interface (Fig. 4.1) that describes the individual steps together with a CAD-like GUI for the mating operations. A preliminary user study [4] compared the classical programming approach (teach pendant and robot programming language PDL2) with this intuitive programming approach using a subset of the gearbox assembly (four parts and three assembly steps).

¹<http://www.roboticslibrary.org/>

Table 4.3: Evaluation of time taken to program an assembly process

Process step	Robotics expert	Robotics expert
	Our system	Teach pendant
	Time Taken (in min).	Time Taken (in min).
Teaching new assembly task	8	48
Adapting to new object poses	0	23
Reordering assembly steps	2	5

**Figure 4.21:** Execution of the woodworking task using a gantry robot.

A high level description of the assembly steps is shown in Fig. 4.14. The test subject was an expert in using a teach pendant. Using this classical approach, it took him 48 min to program the full assembly, including about 23 min spent on teaching individual poses. Afterwards, he was given a short introduction (10 min) into the new intuitive interface. With this new approach, he required 8 min to program the same task. These results have been summarized in Table 4.3.

4.7.1.2 Woodworking domain

For this evaluation, the application from Section 4.6.2 is used. Fig. 4.21 shows the experimental setup with a Guedel gantry robot. Based on the process flow described in Fig. 4.16, the first step is the assembly of panels to the wooden frame. To do this using our intuitive GUI, the user has to select the wooden frame and panels to be used, and define placing constraints between the panels and the frame. The interface for this step is the same as the one used in the assembly domain. A 3D representation of the assembled models is displayed, on which nailing and sawing lines can be defined. This 3D visualization also offers a simulation of the nailing and sawing lines to check if the parameters were set correctly.

In a preliminary user study we compared the classical programming approach (teach pendant and CAD software) to the intuitive programming approach presented here. The user for the woodworking domain was an expert in using the teach pendant in combination with the CAD software. After a short

4. TASK LEVEL ROBOT PROGRAMMING

Table 4.4: Evaluation of time taken to program a woodworking process involving the construction of a wooden wall

Process step	Domain expert	Domain expert
	Our system	Teach pendant
	Time Taken	Time Taken
	(in min).	(in min).
Teaching new woodworking task	13	47
Reordering process steps	0.2	1.6

introduction of about 10 minutes into the intuitive programming system, the user had to program a wall assembly consisting of picking and placing two panels, nailing along the border of the frame and sawing off the protruding parts of the panels. In total, it took him 47 min for the full assembly with the classical approach using different predefined macros. Compared to this, the same process with our intuitive teaching interface required 13 min minutes. These results are summarized in Table 4.4.

4.7.2 Performance evaluation

In these experiments, we evaluate the different constraint solving approaches that we proposed in the context of real-time robot control. We differentiate between solving geometric constraints only to generate poses in the operational space for the robot end-effector (Table 4.5), and a combination of geometric and environment constraints using iterative and hybrid solvers where the robot’s joint positions are estimated (Table 4.6).

In each of these evaluations, the runtime was averaged over 10000 iterations. All evaluations were performed on a PC with a 4.0GHz Intel Core i7-6700K CPU and 16GB of RAM. Our implementation is based on the NLOpt library, where we set the tolerance of the minimization function to be 10^{-8} and the acceptable tolerances for the inequality constraints as 10^{-6} . The derivative tolerance is set as 10^{-8} .

4.7.2.1 Runtime evaluation for different robotic tasks

To assess the time efficiency of our approach, we evaluated the solver runtimes for each application mentioned in Section 4.5. The evaluation covers a wide variety of tasks having 1–4 constraints and 1–8 degrees-of-freedom in the task nullspace. The constraints include separable and mixed transformation manifolds. The runtimes for the iterative, exact and hybrid solvers presented in Chapter 3 are presented. Since the other frameworks mentioned in Table 4.2 do not completely support the types of constraints used in our applications, they are skipped for this evaluation. The results from this evaluation are summarized in Table 4.5.

From this evaluation it can be clearly seen that in terms of runtime, the exact solver is by far the fastest. The iterative solver is the slowest and the hybrid solver falls in between. This evaluation also shows that as far the runtime is concerned, all three methods are suitable for real-time control applications.

Table 4.5: Runtime evaluation on application scenarios: solving of geometric constraints

Scenario	#Constraints	Nullspace DoF	Iterative Runtime (in ms)	Exact Geometric Runtime (in ms)	Hybrid Runtime (in ms)
A. Plane Distance	1	3	0.6 ± 0.1	0.05 ± 0.01	0.10 ± 0.01
B. Seam welding	1	3	0.8 ± 0.1	0.06 ± 0.01	0.10 ± 0.01
C. Cylinder Grasp	4	1	0.8 ± 0.1	0.06 ± 0.01	0.12 ± 0.01
D. Cup Grasp	4	1	1.2 ± 0.2	0.06 ± 0.01	0.10 ± 0.01
E. Tray Grasp	3	0	2.0 ± 0.2	0.06 ± 0.01	1.10 ± 0.10
F. Tray with object	4	8	N/A	0.17 ± 0.01	–

Evaluations involving environment constraints such as collision avoidance are presented separately in Table 4.6. The evaluation covers a wide variety of tasks having 2–26 task space constraint dimensions and 1–3 priority levels. For the first 5 tasks in Table 4.6, the evaluation was performed by solving the constraints from a random initial pose for the robot. The sixth task in Table 4.6 involves manipulability optimization. In this case, random poses within the reachable range of the robot and on the tabletop were chosen for the object to be picked. The last 4 tasks in Table 4.6 include collision avoidance. In these cases, the colliding object was moved along a pre-defined trajectory. A random trajectory was not chosen for the colliding object since it could lead to situations where collision avoidance is impossible. The trajectory was chosen in a way that the robot was able to avoid the collision while satisfying the geometric constraints imposed by the underlying task.

In this evaluation, the runtime of our iterative approach with COBYLA and SLSQP solvers and the hybrid approach with COBYLA solver were compared. The derivative-based SLSQP solver provides better runtimes than the derivative-free COBYLA solver. It is clear from the evaluation, that the hybrid approach is overall faster than the iterative approach.

In Fig. 4.22 we also present plots of the runtime for the evaluation scenarios in Table 4.6 using our iterative approach with the SLSQP solver. It is clear from the plot that the runtimes are significantly affected by the presence of objects within the minimum collision distance in the scene. Hence, two different averages are presented: one for the case where there collision objects are far away, and the other for cases where collision objects are within the thresholds. Note that although the time spent for collision detection is significantly improved by using a convex decomposition of the scene, it still depends heavily on the complexity of the scene.

Optimizing the runtimes for collision detection is beyond the scope of this thesis. The collision detection time affects the overall runtime of the solver, and it is difficult to separate collision detection time and solver time. Hence, two averages for runtimes are presented in case of applications involving collision detection. One average considers cases where objects are far away and collision detection is not triggered. The second average considers the cases where the collision constraint is active.

4. TASK LEVEL ROBOT PROGRAMMING

Table 4.6: Runtime evaluation on application scenarios: solving of geometric constraints with environment constraints and optimization goals

Scenario	Constraint Dimensions	No. of Priority Levels	Runtime COBYLA (in ms)	Runtime SLSQP (in ms)	Runtime Hybrid (in ms)
Plane Distance	2	1	0.6 ± 0.1	–	0.1 ± 0.01
Cylinder Grasp	5	1	0.8 ± 0.1	0.5 ± 0.1	0.1 ± 0.01
Cup Grasp	6	1	1.2 ± 0.2	0.35 ± 0.1	0.1 ± 0.01
Cup Grasp (LWR)	6	1	–	0.65 ± 0.3	–
Tray Grasping (Dual)	12	2	3.0 ± 0.2	–	–
Pick-Place + Manipulability	7	3	2.5 ± 0.3	–	–
Cup + Collision	12	2	2.3 ± 0.3	0.8 ± 0.3	1.1 ± 0.10
Cylinder + Collision	11	2	–	1.7 ± 0.2	–
Cup + Collision (LWR)	13	2	3.3 ± 0.3	1.6 ± 0.3	2.0 ± 0.10
Tray Hold + Collisions (Dual)	26	2	15 ± 3.0	–	–

4.7.2.2 Convergence properties of different solvers

In this section, we evaluate the convergence properties of different solvers. One of key properties of our exact geometric solver is the ability to generate valid solutions in the constraint nullspace independent of the initialization. This is a key difference from iterative solvers, especially gradient-based solvers where the convergence of the iteration and the generated solution depends on the initialization. We perform a set of experiments to highlight this difference.

For this benchmark, we evaluate our iterative solver based described in Section 3.5 against our hybrid solver described in Section 3.6. We choose a set of robotics tasks from Section 4.5 as examples. The initial pose of the robot is randomly sampled in the robot’s valid joint space \mathbf{q}_{rand} , i.e., within its joint limits. Given this initialization, we use each solver to generate an optimal robot joint configuration \mathbf{q}_{opt} that satisfies the task constraints. We analyze two properties of these solvers: satisfaction of the task constraints in the generated solution, and the time required to generate this valid solution. We perform the evaluation over 10000 samples of robot joint positions.

As described in Section 3.6, the Hybrid solver first uses the exact geometric solver over the geometric constraints in the task to generate a geometric manifold \mathcal{M}_{task} that represents the constraint nullspace. It performs projection or direct sampling on this geometric manifold followed by an Inverse Kinematics step to generate a robot joint configuration \mathbf{q}_{seed} that satisfies the geometric constraints for the task. This is used as the seed value for the iterative solver that optimizes the robot joint pose \mathbf{q}_{opt} over the full set of constraints that describe the task, including environment and robot model constraints.

The iterative solver described in in Section 3.5 directly optimizes the robot joint pose \mathbf{q}_{opt} from the randomly generated starting joint pose \mathbf{q}_{rand} .

Table 4.7: Convergence properties of different solvers

Scenario	Framework	Convergence (%)	Runtime (in ms)
Plane distance	Iterative	79.4	3.07
	Hybrid	93.0	1.06
Cup grasping	Iterative	58.2	6.63
	Hybrid	83.1	2.97
Tray grasping	Iterative	19.2	2.43
	Hybrid	100.0	0.16
Cup grasping (LWR)	Iterative	82.2	8.17
	Hybrid	91.5	3.36
Tray grasping (LWR)	Iterative	24.4	5.04
	Hybrid	99.9	0.20

Table 4.7 summarizes the results of this evaluation. The column **Convergence** presents the percentage of these random samples that yielded a valid solution for the task. The **Runtime** column presents the solver runtime required to generate each solution, averaged over the 10000 samples. It can be clearly seen from this Table that the Hybrid solver exhibits better convergence properties and also a much better runtime compared to the iterative solver. Note that the runtime for the Hybrid solver also includes the time required by the exact solver, and the (potentially) multiple Inverse Kinematics steps required during projection and direct sampling. The Inverse Kinematics steps is a large fraction of the overall runtime for the Hybrid solver. We use an iterative Inverse Kinematics solver based on NLOpt from the Robotics Library [69]. If an analytical form of the Inverse Kinematics is available, this runtime could also be significantly lower.

We also observed in these experiments that the benefits of using the Hybrid solver are much more prominent in more complicated tasks where the task nullspace is smaller, e.g. *Tray grasping* compared with *Plane distance*. When the task nullspace is small (e.g. the *Tray grasping* task), the target for the iterative solver can be far away from the random initialization. This results in low convergence rates and a higher probability of getting stuck in a local minima. In this case, the seed solution obtained from the exact solver step is beneficial to provide a good starting pose to the iterative solver. Also, since the task nullspace is small, the *optimal* solution for the iterative solver can be close to this initialization leading to better convergence properties and improved runtime. In case of tasks where the task nullspace is large and well distributed (e.g. the *Plane distance* task), the average distance of the random initialization to the optimal solution is less leading to better convergence and runtime properties.

Note that the *Cup grasping* task has a mixed transformation manifold as its task nullspace. For this example, the results for the Hybrid and Iterative solvers are closer. This is probably because this task often required more iterations of the direct sampling and Inverse Kinematics steps, which increase

4. TASK LEVEL ROBOT PROGRAMMING

Table 4.8: Runtime evaluation of control frameworks

Framework	Runtime	Runtime	Output
	1 constraint (in ms)	2 constraints (in ms)	
Iterative (GN solver) [65]	3.21 ± 0.70	3.89 ± 0.50	\mathbf{x}
Iterative (NLOpt)	0.54 ± 0.07	0.62 ± 0.08	\mathbf{x}
Exact	0.05 ± 0.01	0.06 ± 0.01	\mathbf{x}
Hybrid	0.10 ± 0.01	0.11 ± 0.01	\mathbf{x}
TSID [40, 41]	0.50 ± 0.10	0.50 ± 0.10	\mathbf{q}
WBCF [28, 41]	0.80 ± 0.10	0.80 ± 0.10	\mathbf{q}
Iterative (GN solver) [65]	4.00 ± 1.00	5.00 ± 0.80	\mathbf{q}
Iterative (NLOpt)	0.80 ± 0.10	0.90 ± 0.10	\mathbf{q}
Exact + IK	0.28 ± 0.03	0.29 ± 0.03	\mathbf{q}

the overall runtime. We also restricted this sampling to 20 iterations and this was often not enough to generate a Cartesian pose for the end-effector which is reachable for the robot. This also highlights a limitation of the exact solver that it always generates a valid pose in the Cartesian space but doesn't consider whether the pose is reachable for the robot in terms of its joint limits. In spite of the higher amount of time spent on this sampling step, the resulting seed solutions are effective in improving the convergence and runtime properties of the following iterative solver step.

We also included experiments with the 7 DoF Kuka LWR robot. Generally, the convergence rates for both solvers are better for this robot. This is due to the extra 7 DoF kinematic structure that provides more room for optimization for the iterative solver. However, the difference in runtime is more significant in this case. This highlights the importance of good initializations for the iterative solver, especially in the case of redundant robots and higher-dimensional joint spaces.

4.7.2.3 Comparison with other constraint-based approaches

Based on reference implementations from our previous works [41], we compare our proposed approaches to TSID [40] and WBCF[28] in terms of average controller runtime and task errors. The first test case involves a *plane-plane-coincident* constraint where 3 DoF are fixed. The second test case has two *plane-plane-coincident* constraints, making 5 DoF fixed. The tolerance for task errors is set to 10^{-6} . The results are summarized in Table 4.8.

It should be noted that the frameworks TSID and WBCF generate robot torques $\boldsymbol{\tau}$ (Table 4.2) and a forward dynamics step was added in our implementation [41] to obtain robot joint positions \mathbf{q} . [65] and [7] generate robot joint positions \mathbf{q} , but we modified our implementations so that they also generate only target Cartesian positions \mathbf{x} . This is to ensure a fair comparison with the Exact solver that generates

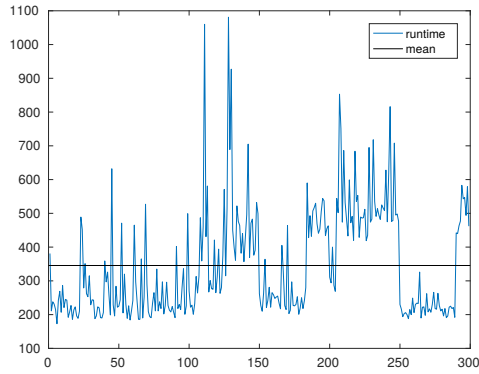
Cartesian positions \mathbf{x} . In this evaluation, we used the Inverse Kinematics (IK) implementation from [69] to generate robot joint positions \mathbf{q} from Cartesian positions \mathbf{x} provided by the exact solver.

The timing for generating robot joint positions including the additional IK step has been reported separately for this evaluation. There are two reasons for this. Firstly, many controllers directly support input in the form of end-effector Cartesian positions. Secondly, runtimes for IK can vary immensely depending on the kinematics of the robot and the IK algorithm (symbolic or iterative).

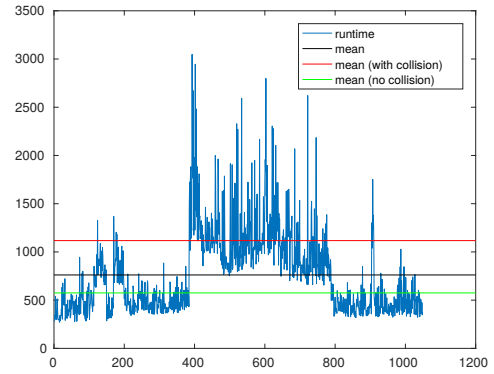
4.8 Discussion

In this Chapter, we presented the primary application of this thesis, i.e, a constraint-based robot motion control framework. We presented our approach for realizing our objective of creating an intuitive, task-level programming interface for robots. We demonstrated how a constraint-based definition of common robotic tasks is not only intuitive for the user, but also efficient in terms of the programming time. We also showed how task-level constraints can be combined with environment constraints during runtime. Through several evaluations, we have shown that the runtime performance of our constraint solvers is suitable for real-time control.

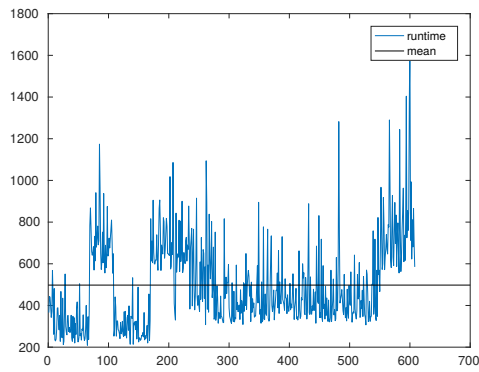
4. TASK LEVEL ROBOT PROGRAMMING



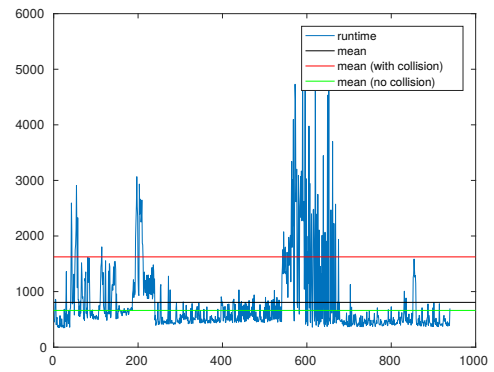
(a) Cup grasping from the side



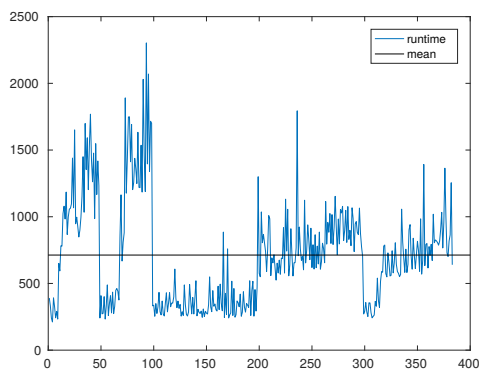
(b) Cup grasping from the side + collisions



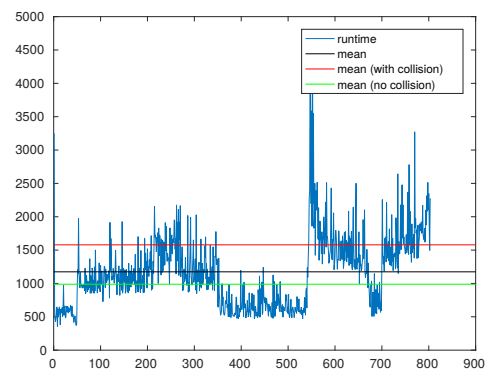
(c) Cylinder grasping from the rim



(d) Cylinder grasping from the rim + collision



(e) Cup grasping from the side (LWR)



(f) Cup grasping from the side + collisions (LWR)

Figure 4.22: Runtime plots for tasks in Table 4.6.

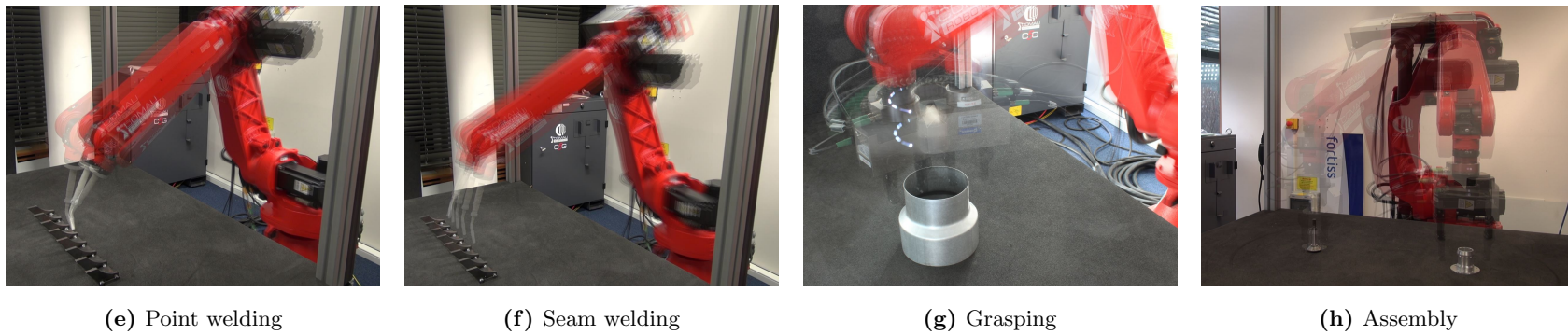
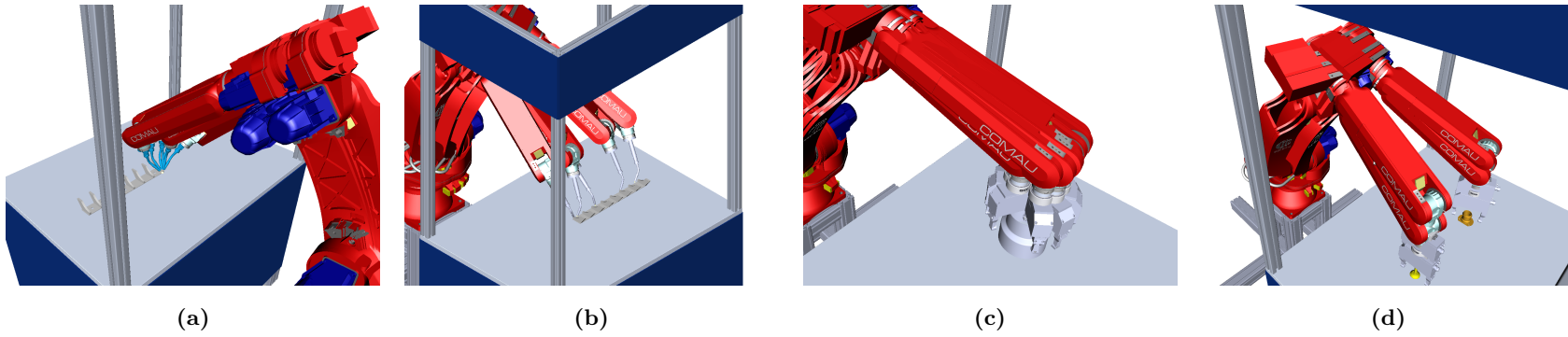
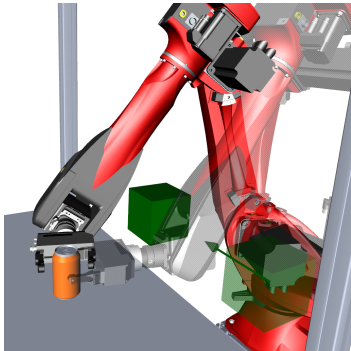
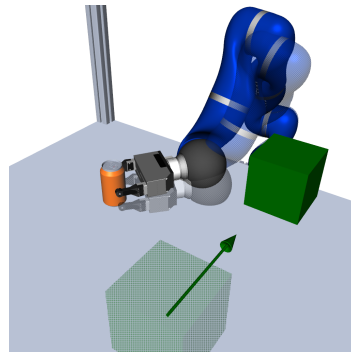


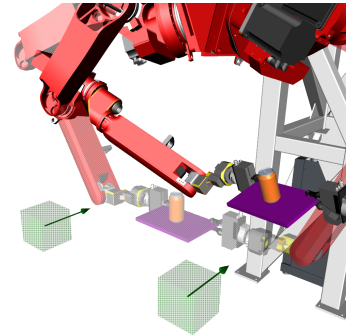
Figure 4.23: Execution of constraint-based tasks in a robotic workcell. CAD semantics allow these tasks to be defined in terms of geometric constraints between features of robot tools and objects, allowing intuitive robot programming.



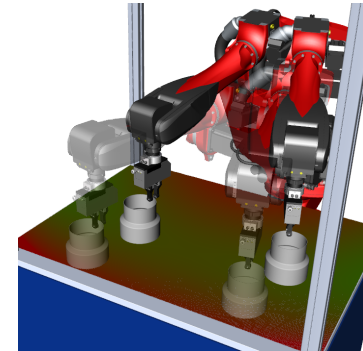
(a) Cup grasping with obstacle avoidance using 6 DOF Comau RACER 7-1.4



(b) Cup grasping with obstacle avoidance using 7 DOF LWR



(c) Grasping a tray with dual arm and avoiding multiple obstacles



(d) Pick cylinder at rim and place with manipulability optimization

Figure 4.24: Execution of constraint-based robotic tasks in the presence of environment constraints. a Cup grasping with obstacle avoidance using 6 DOF Comau RACER 7-1.4, b Cup grasping with obstacle avoidance using 7 DOF LWR, c Grasping a tray with dual arm and avoiding multiple obstacles, d Pick cylinder at rim and place with manipulability optimization.

Chapter 5

Constraint-based Motion Planning

In this Chapter, we present a constrained motion planning algorithm that computes collision free paths for the robot in order to execute a task. The target for the robot is not defined in the form of joint or operational positions, but using geometric constraints. Hence, the target for the robot is not a fixed pose but a geometric manifold. Additionally, we can define constraints on the path to be followed by the robot. As an example, a robot carrying a tray with an open cup containing liquid can be constrained to keep the tray horizontal while moving in order to ensure that the liquid is not spilled. Our algorithm computes a set of waypoints for the robot trajectory such that each waypoint satisfies the path constraint and the final target lies on the goal manifold.

5.1 Introduction

The definition of robotic tasks using constraint-based approaches is already popular in literature, and also a core focus of this thesis. Constraint-based methods have proven to be powerful, efficient and intuitive [4, 24] for this purpose. The constraints in question can arise from the task description itself, e.g. geometric constraints involving objects to be manipulated and the robot's body. The environment in which the robot operates also poses certain constraints, e.g. obstacles. The robot's kinematic and dynamic properties also constrain its motion. As shown in Chapters 2 and 4, all these constraints can be modeled in a unified constraint-based control framework. The constrained motion planning problem involves generating optimal motion trajectories for the robot while following these constraints.

The constraint-based control framework presented in Chapter 4 focuses on the formulation of robotic tasks using constraints. The constraint solver operating on these constraints can either be for offline programming, or for real-time reactive control. This Chapter describes a motion planning approach based on constraint-based task descriptions. While the controller in Chapter 4 can be used for reactive control and trajectory adaptation, it is not suitable for generating complete trajectories.

In this work, we present a generic constrained motion planning formulation that can be applied to any classical sampling-based motion planning algorithm such as RRT [70], RRT Connect [71], EET [72], etc. In this work, we use the EET algorithm as an example, and modify it to plan in the nullspace of the task constraints.

5.2 Related Work

Our approach consists of four main components: intuitive robot task representation using constraints, an exact geometric solver, a general sampling based planning algorithm and a constraint-based robot motion controller.

Our modeling approaches for representing robot tasks using constraints is described in Chapter 2. We have presented many examples of complex robotic tasks described using different types of constraints such as geometric, robot model and environment constraints in Chapter 4. We have shown how such constraint-based descriptions are intuitive and powerful. The exact geometric solver generates transformation manifolds from the geometric constraints in the task description. These transformation manifolds help restrict the search space of the motion planner. Projection operators, distance functions and parametric equations defined for these transformation manifolds enable us to easily adapt sampling-based motion planning algorithms to constrained motion planning algorithms.

In Chapter 4, we presented a constraint-based robot control approach. We presented several examples that involve complex robotic tasks and included geometric as well as environment constraints such as obstacle avoidance. In general, constraint-based motion control approaches such as those based on nullspace projections [28], task-space inverse dynamics [40, 41], solver based approaches [7, 65], or exact approaches [6] all focus on *control* that generates the next configuration based on the current state. While the approaches using different flavors of nullspace projections usually focus on local problems, optimization-based approaches also have a limited time horizon. It is easy for such approaches to get stuck in local minima in the presence of environment constraints such as obstacles. Hence, global planning is essential for manipulation tasks.

A number of sampling-based global planning approaches have been developed to efficiently search high-dimensional C-spaces [73]. Such sampling-based planners are designed to be efficient in exploring the solution space, without needing the exhaustive computational requirements of dynamic programming or getting stuck in local minima as seen in gradient-descent based methods typically used in optimization-based controllers. On the other hand, control approaches are suitable for high-speed dynamic adaptation to ensure that the task is executed in uncertain environments.

We try to combine the advantages of both approaches by using them at different levels. The constraint-based descriptions of the goal and path are used for the motion planning task to generate a trajectory. Once this trajectory is computed, the motion controller is used for real-time control of the robot. Following this trajectory is then added as a constraint to the set of task constraints for the motion controller. The trajectory following task is kept at a lower priority level than the task constraints. This ensures that dynamic adaptations required due to uncertain environments don't violate the task constraints but may affect the pre-planned trajectory.

Overall, our approach is able to use intuitive and powerful robot task descriptions expressed as constraints to perform efficient, constrained motion planning and real-time constrained motion control.

5.2.1 Contribution

Robot motion planning using constraints on the pose of the end-effector has been used in several works such as Koga et al. [74], Yamane et al. [75], Yao et al. [76], Bertram et al. [77], Drumwright et al. [78], Stilman et al. [79], and Berenson et al. [80]. Although often able to solve the problem at hand, these

algorithms suffer from problems with efficiency [79], probabilistic incompleteness ([74], [75], [76]), and overly-specialized constraint representations ([77], [78], [80]).

An important concept in our work is the use of an exact geometric solver to generate task nullspaces. Several works have proposed the use of geometric subspaces that restrict the sampling spaces for motion planners. Our approach involves automatic generation of these geometric subspaces from constraint-based task descriptions. The exact geometric solver developed in this thesis supports non-linear geometric constraints with inequalities. It can also handle constraint combinations where the resulting nullspace has mixed degrees of freedom, i.e. rotation and translation elements cannot be controlled separately. Some of these constraint nullspaces are often very complicated and difficult to visually describe or formulate. Hence, approaches such as controlled or free axes [79], or Task Space Regions [80] cannot model tasks involving such nullspaces. On the other hand, the geometric nullspace definitions formulated by these approaches can also be formulated using our constraint modeling approach. Also, modeling the task nullspace region as a constraint is relatively easy compared to the reverse, i.e. manually calculating the nullspace from a set of constraints.

5.3 Motion Planning Approach

Our extensions to classical motion planning algorithms are rather generic and can be applied to several approaches. As an example, we use the EET motion planner [72] as the reference. We highlight the adaptations done to this algorithm in order to make it compliant to constraint-based task definitions.

Many sampling-based algorithms use a random sampling strategy to generate a random position in the robot’s operational or configuration space, or sample the goal itself. In the EET sampling algorithm, the goal is sampled as the next sample with a specific probability. In our constraint-based adaptation, the goal for the motion planner is defined using a set of geometric constraints $\mathcal{M}_{\text{goal}}$ rather than a fixed operational position \mathbf{T}_{goal} . Hence, this sample is a projection of the nearest operational position in the tree \mathcal{G} onto the goal manifold.

Once the sample $\mathbf{T}_{\text{sample}}$ is generated, the graph \mathcal{G} is extended towards it. In our constraint-based extension, this sample is additionally projected onto the path constraints $\mathcal{M}_{\text{path}}$ to ensure that the path constraints are satisfied.

The motion planner terminates when the extended vertex satisfies the goal constraint, i.e., its distance from the goal manifold is within a specified threshold.

Our constraint-based EET approach is defined in Algorithm 5, and its steps are explained in the following sections.

This approach can also be extended for planners with two trees: one growing from the initial position, and another from the goal. Since the goal is defined as a geometric manifold, there can be multiple vertices representing the goal state.

5.3.1 Sampling

The probabilistic planning approach that we use is based on random sampling to generate candidate configurations. For the constrained planning approach, it is necessary that these candidates lie in the

5. CONSTRAINT-BASED MOTION PLANNING

Input: $q_{\text{start}}, \mathcal{M}_{\text{goal}}, \mathcal{M}_{\text{path}}$
Output: $G = (V, E)$
 $V \leftarrow q_{\text{start}}$ *tree initialization with start configuration*
 $E \leftarrow \emptyset$
 $T_{\text{goal}} \leftarrow \text{PROJECT}(q_{\text{start}}, \mathcal{M}_{\text{goal}})$
 $q_{\text{goal}} \leftarrow \text{IK}(T_{\text{goal}})$
 $\sigma \leftarrow \gamma$ *initialize exploration/exploitation balance*
repeat *search until goal reached*
 $(q_{\text{near}}, T_{\text{sample}}) \leftarrow \text{SAMPLE}(G, \sigma, \mathcal{M}_{\text{goal}}, \mathcal{M}_{\text{path}}, T_{\text{goal}})$ *sample new workspace frame and select nearest vertex*
 $(q_{\text{new}}, T_{\text{new}}) \leftarrow \text{CONNECT}(q_{\text{near}}, T_{\text{sample}}, \mathcal{M}_{\text{path}})$
 if $q_{\text{new}} \neq \emptyset$ **then**
 $V \leftarrow V \cup q_{\text{new}}$
 $E \leftarrow E \cup (q_{\text{near}}, q_{\text{new}})$
 $\sigma \leftarrow (1 - \alpha)\sigma$ *increase exploitation*
 end
 else *expansion unsuccessful*
 $\sigma \leftarrow (1 + \alpha)\sigma$ *decrease exploitation*
 end
 if $\sigma > 1$ **then** *perform backtracking to prev. sphere*
 $s \leftarrow s_{\text{parent}}$ *select previous sphere*
 $\sigma \leftarrow \gamma$ *reset exploration/exploitation balance*
 end
until $\|\text{PROJECT}(T_{\text{new}}, \mathcal{M}_{\text{goal}}) - T_{\text{new}}\| < \epsilon$
return G *configuration space tree*

Algorithm 5: Constraint-based EET algorithm

nullspace of the path constraints. There are three general categories of such sampling approaches: rejection, projection, and direct sampling [80].

The choice of sampling algorithm depends on the task definition, and the task nullspace. In cases where the manifold is a large space, e.g. a plane representing the tabletop, projection sampling strategy is useful. This is because the projection operation can generate a configuration that is closest to the nearest vertex in the tree. However, in cases where the manifold is a small and restricted space, the projection sampling strategy is not as effective as the direct sampling strategy.

5.3.1.1 Rejection Sampling

In this sampling method, we simply generate random samples in the robot joint space q_{sample} and check if the corresponding operational position T_{sample} satisfies the path constraints $\mathcal{M}_{\text{path}}$. If the constraints are satisfied, this is considered a valid sample and discarded otherwise. This sampling strategy is useful in cases where the constraint nullspace is rather large (occupies a large fraction of the operational space) and there is a high probability that random sampling generates a valid sample. The benefit of this approach is that only a binary in-out function need to be defined for the constraints.

```

Input:  $G, s, \sigma, \mathcal{M}_{\text{goal}}, \mathcal{M}_{\text{path}}, \mathbf{T}_{\text{goal}}$ 
Output:  $\mathbf{q}_{\text{near}}, \mathbf{T}_{\text{sample}}$ 
if RAND() <  $\rho$  then sample goal
     $\mathbf{p}_{\text{sample}} \leftarrow \mathbf{p}_{\text{goal}}$  select goal position
     $\mathbf{R}_{\text{sample}} \leftarrow \mathbf{R}_{\text{goal}}$  select goal orientation
     $\mathbf{T}_{\text{proj}} \leftarrow \text{PROJECT}(\mathbf{T}_{\text{sample}}, \mathcal{M}_{\text{path}})$ 
     $\mathbf{q}_{\text{near}} \leftarrow \text{NEAREST}(G, \mathbf{T}_{\text{proj}})$  nearest vertex in tree
else expansion unsuccessful
     $\mathbf{p}_{\text{sample}} \leftarrow \mathcal{N}(\mathbf{p}_s)$  sample near last point
     $\mathbf{R}_{\text{sample}} \leftarrow \mathcal{U}()$  uniform sampling for orientation
     $\mathbf{T}_{\text{proj}} \leftarrow \text{PROJECT}(\mathbf{T}_{\text{sample}}, \mathcal{M}_{\text{goal}})$ 
     $\mathbf{q}_{\text{near}} \leftarrow \text{NEAREST}(G, \mathbf{T}_{\text{proj}})$  nearest vertex in tree
end
return  $(\mathbf{q}_{\text{near}}, \mathbf{T}_{\text{proj}})$ 
    
```

Algorithm 6: SAMPLE algorithm

5.3.1.2 Direct Sampling

In this approach, we generate samples directly in the constraint nullspace. We use a parametric representation of the constraint nullspace, and perform random sampling in the space of these parameters. This approach guarantees that the generated samples are always valid and conform of the task constraints. The tricky aspect in this sampling approach is the mapping between the parametric space and task space. For motion planning, it is often required to generate samples close to the current ones. However, samples that are close in the parametric space may not be close in the operational or task space.

5.3.1.3 Projection Sampling

The sampling approach is described in Algorithm 6. Random samples are generated in the operational position $\mathbf{T}_{\text{sample}}$ of the robot. $\mathbf{T}_{\text{sample}}$ is then projected into the nullspace of the path constraints $\mathcal{M}_{\text{path}}$ using the projection function $\text{PROJECT}(\mathbf{T}_{\text{sample}}, \mathcal{M}_{\text{path}})$. The corresponding robot joint position \mathbf{q}_{proj} is obtained using an inverse kinematics function IK. Similarly, the goal manifold can be sampled by projecting the random sample $\mathbf{T}_{\text{sample}}$ onto the goal manifold $\mathcal{M}_{\text{goal}}$.

In this work, we use the projection sampling approach due to the following reasons:

- Our sampling space is rather low-dimensional (6D operational space). Hence, sampling time is not a big issue.
- We can directly use existing planning algorithms without much changes. Only a projection function in the operational space needs to be specified.
- The main bottleneck of projection sampling is distance computation from the manifold. With our exact solver, this is very fast. The computation time for this will be too high if an iterative solver is used instead.

5. CONSTRAINT-BASED MOTION PLANNING

Input: $\mathbf{q}_{\text{near}}, \mathbf{T}_{\text{sample}}, \mathcal{M}_{\text{path}}$
Output: $\mathbf{q}_{\text{new}}, \mathbf{T}_{\text{new}}$
if SINGULAR(\mathbf{q}_{near}) **then** *within singularity*
 | $\mathbf{q}_{\text{new}} \leftarrow \text{RAND}()$ *uniform sampling for singularities*
else
 | $\Delta \mathbf{x} \leftarrow \mathbf{T}_{\text{near}} - \mathbf{T}_{\text{sample}}$
 | $\Delta \mathbf{q} \leftarrow \mathbf{J}^\dagger(\mathbf{q}_{\text{near}})\Delta \mathbf{x}$
 | $\mathbf{q}_{\text{new}} \leftarrow \mathbf{q}_{\text{near}} + \Delta \mathbf{q}$
 | $\mathbf{T}_{\text{new}} \leftarrow \text{FK}(\mathbf{q}_{\text{new}})$
 | $\mathbf{T}_{\text{proj}} \leftarrow \text{PROJECT}(\mathbf{T}_{\text{new}}, \mathcal{M}_{\text{path}})$
 | $\mathbf{q}_{\text{proj}} \leftarrow \text{IK}(\mathbf{T}_{\text{proj}})$
end
if $\mathbf{q}_{\text{proj}} \in \mathcal{C}_{\text{free}}$ **then**
 | **return** $(\mathbf{q}_{\text{proj}}, \mathbf{T}_{\text{proj}})$
else
 | **return** (\emptyset, \emptyset)
end

Algorithm 7: EXTEND algorithm

5.3.2 Extension

The sampling step provides the generated random sample in the operational space $\mathbf{T}_{\text{sample}}$, along with the nearest node the existing tree G (joint position \mathbf{q}_{near} and operational position \mathbf{T}_{near}). The EXTEND step then generates a point \mathbf{T}_{new} between \mathbf{T}_{near} and $\mathbf{T}_{\text{sample}}$ based on the obstacles and potential singular configurations. In the presence of path constraints $\mathcal{M}_{\text{path}}$, the generated point is projected onto the constraint nullspace to generate a new point \mathbf{T}_{proj} , which is added to the tree G . This step is described in Algorithm 7.

Note that the collision checking step ($\mathbf{q}_{\text{proj}} \in \mathcal{C}_{\text{free}}$) is the most expensive part of this algorithm. Also, this algorithm includes an inverse kinematics step ($\mathbf{q}_{\text{proj}} \leftarrow \text{IK}(\mathbf{T}_{\text{proj}})$) which can be expensive in case of an IK algorithm using iterative solvers. This step can be sped-up if an analytical form of inverse kinematics is available.

5.3.3 Connect

The sampling step provides the generated random sample in the operational space $\mathbf{T}_{\text{sample}}$, along with the nearest node the existing tree G (joint position \mathbf{q}_{near} and operational position \mathbf{T}_{near}). Note that the EXTEND algorithm only moves one *step* ($\Delta \mathbf{q}$) from the nearest vertex in the tree towards the sample. The CONNECT step incrementally *extends* the nearest vertex in the tree towards the sample until it hits an obstacle. This algorithm calls the EXTEND function in a loop until the vertex cannot be further extended towards the sample. Algorithm 8 explains this approach.

```

Input:  $q_{\text{near}}, T_{\text{sample}}, \mathcal{M}_{\text{path}}$ 
Output:  $q_{\text{new}}, T_{\text{new}}$ 
 $q_{\text{new}} \leftarrow \emptyset$ 
 $T_{\text{new}} \leftarrow \emptyset$ 
repeat
   $(q_{\text{new}'}, T_{\text{new}'}) \leftarrow \text{EXTEND}(q_{\text{near}}, T_{\text{sample}}, \mathcal{M}_{\text{path}})$ 
  if  $q_{\text{new}'} \neq \emptyset$  then
     $q_{\text{near}} \leftarrow q_{\text{new}'}$ 
     $q_{\text{new}} \leftarrow q_{\text{new}'}$ 
     $T_{\text{near}} \leftarrow T_{\text{new}'}$ 
     $T_{\text{new}} \leftarrow T_{\text{new}'}$ 
  end
until  $q_{\text{new}'} = \emptyset$ 
return  $(q_{\text{new}}, T_{\text{new}})$ 

```

Algorithm 8: CONNECT algorithm

5.4 Experiments

In this section, we present some constrained motion planning examples that are based on common robotic tasks. We specify the goal and path constraints for each of these tasks and show the paths and waypoints calculated by our constrained motion planning algorithm.

5.4.1 Moving a tray carrying an object

In this example, the robot end-effector grasps a tray on which a soda can is placed. The task for the robot is to move from a start point to an end point while keeping the tray upright in order to prevent the object from sliding off the tray. The constraint-based description of this task is presented in Section 4.5.2. The Plane_Y , that is a plane parallel to the Y direction of the gripper grasping the tray, needs to be parallel to the Plane_{A1} of the tray as shown in Fig. 4.7. The path constraint $\mathcal{M}_{\text{path}}$ for this task is:

$$\text{Parallel}(\text{Plane}_Y, \text{Plane}_{A1}) \quad \} \quad \mathcal{M}_{\text{path}} \quad (5.1)$$

The planned path, in the presence of a cube-shape obstacle in the scene, and intermediate poses calculated by the constrained EET motion planner are shown in Fig. 5.1.

5.4.2 Seam welding with start and end point

This task involves two steps. The first step is to reach the start point of the weld seam with an orientation within the acceptable limits for the weld quality. The second step is to move along the weld seam with a constant velocity until the end point is reached. The constraint-based description of this task is defined in Section 4.5.1. For the motion planning problem, the goal for the first step $\mathcal{M}_{\text{goal},1}$ is defined by the constraints in (5.2):

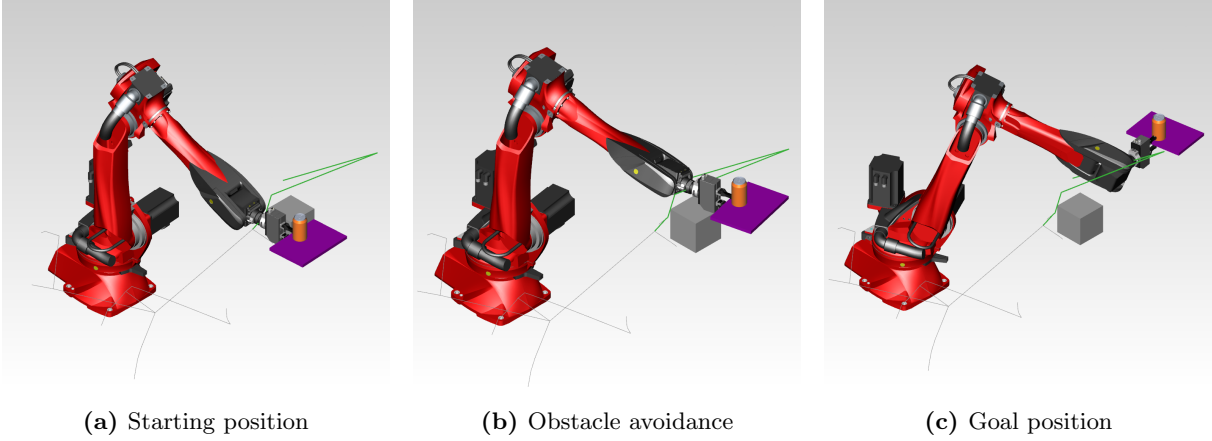


Figure 5.1: Constrained Motion Planning example: Moving a tray while keeping it horizontal

$$\left. \begin{array}{l} 0 \leq \text{Angle}(\text{Axis}_B, \text{Axis}_A) \leq \theta_{\max} \\ \text{Coincident}(\text{Point}_B, \text{Point}_{A1})(\text{start point}) \end{array} \right\} \mathcal{M}_{\text{goal},1} \quad (5.2)$$

The goal for the second step $\mathcal{M}_{\text{goal},2}$ is defined by the constraints in (5.3)

$$\left. \begin{array}{l} 0 \leq \text{Angle}(\text{Axis}_B, \text{Axis}_A) \leq \theta_{\max} \\ \text{Coincident}(\text{Point}_B, \text{Point}_{A2})(\text{end point}) \end{array} \right\} \mathcal{M}_{\text{goal},2} \quad (5.3)$$

The path for the second step $\mathcal{M}_{\text{path},2}$ is restricted by the constraints in (5.4)

$$\left. \begin{array}{l} 0 \leq \text{Angle}(\text{Axis}_B, \text{Axis}_A) \leq \theta_{\max} \\ \text{Coincident}(\text{Point}_B, \text{Line}_A) \\ \text{Velocity}(\text{Point}_B) \cdot \text{Axis}_A = 0.1\text{m/s} \end{array} \right\} \mathcal{M}_{\text{path},2} \quad (5.4)$$

The planned path and intermediate poses calculated by the constrained EET motion planner are shown in Fig. 5.2.

5.4.3 Grasping a tray by its handle

In this example, the task for the robot is to grasp a tray by its handle using a two-finger gripper. The robot moves from a start position to a grasping position for the tray handle. The tray can be grasped at any point along its handle. Since the two-fingered gripper also automatically centers the tray, there is an additional nullspace along the length of the finger and also along the opening width of the gripper. The constraint-based description of this task is similar to that of the task presented in Section 4.5.4. Instead of the Plane-Plane-Distance-Min-Max constraint, this task requires a Line-Line-Distance-Min-Max constraint (5.5).

The planned path and Intermediate poses calculated by the constrained EET motion planner are shown in Fig. 5.3.

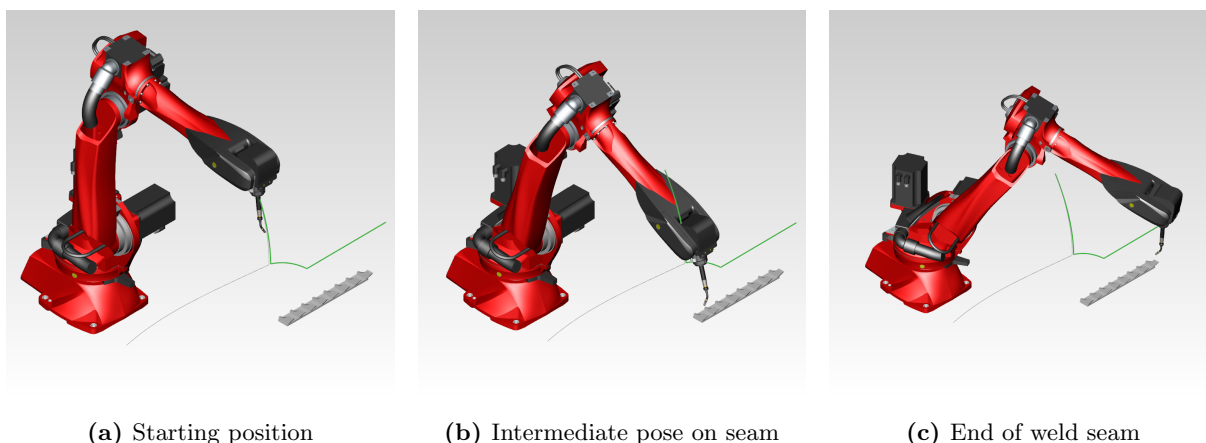


Figure 5.2: Constrained Motion Planning example: A seam welding task

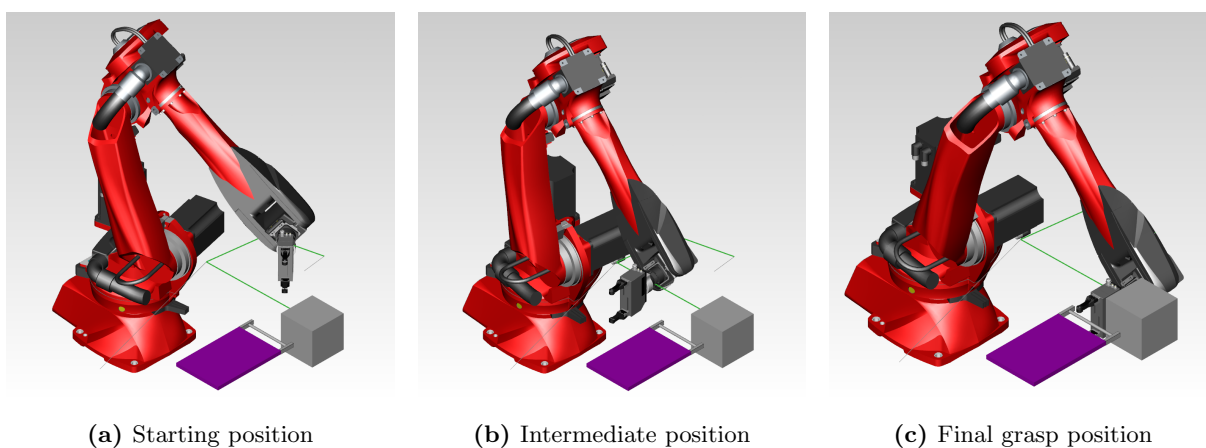


Figure 5.3: Constrained Motion Planning example: Grasping a tray by its handle

$$0 \leq \text{Distance}(\text{Line}_{A_2}, \text{Plane}_{B_2}) \leq \min(L_f/2, W_f/2) \quad \left. \vphantom{\text{Distance}} \right\} \mathcal{M}_{\text{goal}} \quad (5.5)$$

5.5 Extensions

While this Chapter has focused on the constrained motion planning algorithm itself, there are several extensions possible to this approach in order to integrate it with perception or motion control algorithms.

5.5.1 Combination with constraint-based motion controller

The motion planner can be combined with the constraint-based real-time robot motion controller described in Chapter 4. The path and goal constraints are added to the motion controller, along with the environment and robot model constraints. The trajectory calculated by the motion planner acts as a low-priority constraint for the motion controller. The motion controller ensures that the task constraints are satisfied in the presence of environmental uncertainties, e.g. moving objects and slight changes in the scene.

5.5.2 Including uncertainties

Our motion planning and control approach can be extended to account for uncertainties in the object’s position. Our constraint-based pose estimation algorithm (Chapter 6) provides not only the underspecified pose of the object but also pose uncertainties in different axes or w.r.t the primitive shape geometries that form the object. We incorporate this uncertainty in the form of inequalities in the task constraints.

As an example, consider the uncertainties in the pose of the welding workpiece in Section 5.4.2. We project the uncertainties in the object’s pose (say ϵ) onto the line defining the weld seam (ϵ_A), and to the start and end points of the seam (ϵ_{A1} and ϵ_{A2} respectively).

The equality constraints that define the task are then adapted to corresponding inequality constraints 5.6.

$$\begin{aligned} \epsilon_A &\leq \text{Distance}(\text{Point}_B, \text{Line}_A) \leq \epsilon_A \\ \epsilon_{A1} &\leq \text{Distance}(\text{Point}_B, \text{Point}_{A1}) \leq \epsilon_{A1} \text{ (start)} \\ \epsilon_{A2} &\leq \text{Distance}(\text{Point}_B, \text{Point}_{A2}) \leq \epsilon_{A2} \text{ (end)} \end{aligned} \tag{5.6}$$

5.6 Discussion

In this Chapter, we presented a robotics application of our constrained solving framework in the field of constrained motion planning. We modeled the goal for the motion planner using a set of geometric constraints instead of a fixed pose in the robot’s operational or configuration space. We also added geometric constraints to the path of the robot while moving in its operational space. Our exact geometric solver converts these geometric constraints into transformation manifolds. The distance functions, projection operations and parameteric definitions of these geometric manifolds enable different sampling and termination strategies for the constrained motion planning problem. We showed how a state-of-art sampling based motion planner can be easily adapted to handle a constrained motion planning problem.

Part III

Applications in Computer Vision

Chapter 6

Constraint-based Object Detection and Pose Estimation

In this Chapter, we present a computer vision application of our constraint-based modeling techniques. We model the classical problem of object recognition and pose estimation using geometric constraints. We derive the algorithms for using this approach in the presence of noisy sensor data, partial views, occlusions, and symmetrical objects.

6.1 Introduction

Object recognition and pose estimation is a classic problem in computer vision. Color and shape are the most common forms of visual information used for this task.

In the context of this work, object detection and pose estimation refer to a frame-less system where no prior or temporal information about the pose or even existence of objects in the scene is available. The task is to detect all instances of a known set of objects in a given scene, along with their poses.

Our work can be classified as a model-based detection approach. We assume that the complete CAD model of the object is available a priori. In cases where the object model is only available as a mesh or point cloud, information about primitive shapes in the model can be extracted using our primitive shape decomposition algorithm.

With recent advancements in 3D sensing, purely shape-based methods for object detection and pose estimation are becoming more popular.

6.2 Related Work

This work is focused on detection of industrial objects/workpieces that are typically texture-less. Hence, we focus on purely shape based approaches.

The modeling of the object's geometry is an important aspect to classify the different approaches for this problem. Some classical approaches consider the object model as a set of 3D points. There are two main categories of approaches for this object model. The first category depends on keypoints and feature descriptors associated with them. Some previous works in this direction include local shape

6. CONSTRAINT-BASED OBJECT DETECTION AND POSE ESTIMATION

keypoint [81], spin images [82], Fast Feature Point Histogram (FPFH) [83], interest points [84], BOLD features [85], Implicit Shape Models (ISM) [86], Unique Shape Context (USC) [87] and VFH [88]. A recent survey of these approaches based on implementations available in the Point Cloud Library¹ was presented in [89]. The second category uses randomly selected points on the object model instead of keypoints, and creates a dense set of feature vectors which can be matched to random points in the scene. Previous works using this type of approach include surflet pair based approaches [90, 91, 92], and the triple-point feature method [93]. Each of these approaches have their own advantages and disadvantages. Keypoint-based methods rely on the availability of unique and repeatable points on the object model. The advantage is the computational speed since keypoint detection is usually efficient and the matching phase involves few (key) points. Global descriptors such as VFH [88] require a cumbersome training phase, where a large number of object views need to be generated by real experiments. Besides, their accuracy decreases significantly in case of occlusions and partial views. The advantage of these methods lies in their computational speed. On the other hand, methods such as [91, 92, 93], are designed to be robust to partial views, occlusions and noisy data but don't scale well for real-time applications or object models with a large number of points.

Another way of modeling objects is using information about shapes that comprise the object, i.e., a primitive shape approximation of the object. This model is less generic than points, and is designed to work only for objects that can be accurately approximated using a small set of primitive shapes. The advantage of this modeling approach is that the complexity and size of the model is much smaller compared to that with points. An important property of the typical industrial parts that we deal with, is that they are often composed of simpler geometric parts. Hence, their geometries can be approximated accurately using a set of primitive shapes such as planes, cylinders, spheres, etc. A key idea in our work is to utilize the mathematical properties of such geometric primitives for object detection, since they can be detected more accurately and efficiently than complete CAD models. We use a boundary representation (BREP) of CAD models, since it is a standardized format that retains the information about primitive geometric shapes. Also, BREP is a standard supported by several popular CAD modeling softwares (e.g. SolidWorks) and exchange formats (e.g., STEP, IGES). In [66], we presented an ontological representation of BREP-based CAD models, which we also use in this work. Previous works focusing on efficient methods for primitive shape detection and their applications in object detection and pose estimation were presented by Schnabel [94, 95, 96]. Applications of primitive shape-based object recognition in grasping scenarios were shown in [97, 98].

An approach for efficient object detection and pose estimation, where information about primitive shapes was used in combination with surflet pairs to improve the runtime and accuracy of pose estimation, was presented in our previous work [90].

In [90], we used an energy minimization approach for decomposing object models (in point cloud format) into primitive shapes. This approach discarded the information about primitive shapes contained in the CAD models (stored in STEP, IGES files) and tried to estimate it again, possibly introducing errors in the process. In this work, we directly use the information of primitive shapes stored in CAD models, thereby achieving a perfect decomposition of the CAD models into primitive shapes.

In [90], we presented the concept of object detection using sets of primitive shapes that are minimal sets for 3D pose estimation. However, possible minimal combinations such as 3 Planes or a Plane and

¹www.pointclouds.org

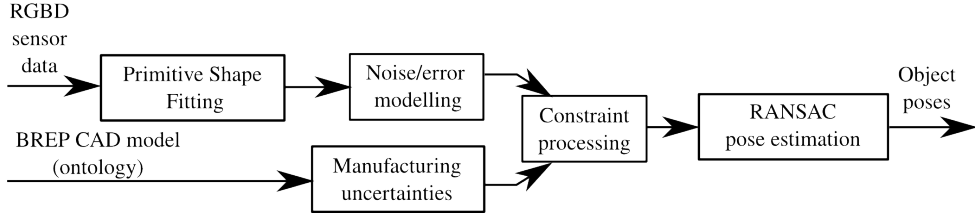


Figure 6.1: Pipeline for primitive shape based object detection from noisy sensor data

a Cylinder were defined explicitly. In this work, we model the constraints that each primitive shape enforces on the object’s pose and can detect these minimal sets automatically from a set of shapes. Our pose estimation approach can handle over-specified constraints and optimize the pose in a least-squares sense.

For estimating object poses, we propose the use of a non-linear least squares solver that supports inequality constraints. The constraints introduced by primitive shape matching are modified to be inequality constraints, since this enables explicit modeling of sensor noise in RGBD data and errors in fitting of primitive shapes to scene point clouds in the pose estimation process in the form of bounds for these inequality constraints. Hence, instead of matching primitive shapes exactly we explicitly model and allow tolerances, which leads to a better convergence of the pose estimation process.

6.3 Object Detection and Pose Estimation Pipeline

Fig. 6.1 presents an overview of our object detection and pose estimation pipeline. In the offline phase, the primitive shape information is extracted directly from the BREP model. Manufacturing uncertainties are also modeled offline. In the live scene, primitive shapes are fit to every frame of 3D sensor data. This step also provides information about sensor noise and shape fitting errors. A matching algorithm creates associations between the scene and model primitive shapes. Each association creates a constraint on the pose of a model and its possible instance in the scene. Using minimal cliques of primitive shapes in a RANSAC-based algorithm, object instance and pose hypotheses are generated. A hypothesis verification step then filters the hypotheses and generates the final instances of objects in the scene along with their poses.

6.4 Primitive Shape Detection

The primitive shape detection pipeline is shown in Fig. 6.2. The point cloud PC is represented as a set of primitive shapes \mathcal{P}_i containing points $PC_i \subseteq PC$ such that $\cup PC_i \subseteq PC$. In this work, the primitive shapes \mathcal{P}_i can be planes, cylinders and spheres. An example of such a decomposition is shown in Fig. 6.3, where the original scene cloud is shown in Fig. 6.3 (a) and its decomposition into primitive shapes is shown in Fig. 6.3 (b).

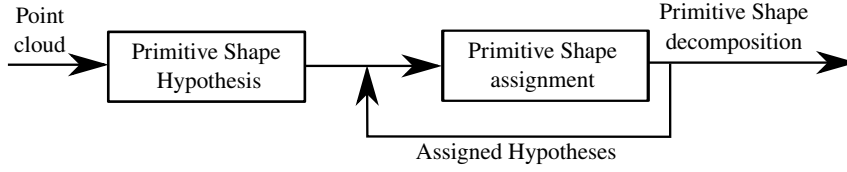


Figure 6.2: Pipeline for Primitive Shape Decomposition

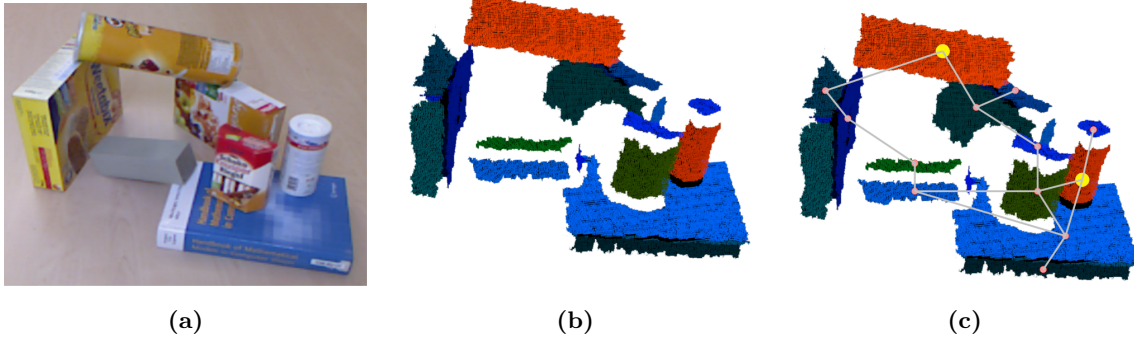


Figure 6.3: Primitive Shape Decomposition example : (a) RGB channel of original Point Cloud (b) result of Primitive Shape Decomposition (c) Primitive Shape Graph representation.

6.4.1 Primitive Shape hypothesis

Hypothesis for primitive shapes are generated by randomly sampling point sets in the point cloud. Once the hypotheses have been generated, each point in the cloud is checked to determine whether it satisfies the hypotheses. The method used for generating a hypothesis and determining its inliers depends on the type of primitive shape.

- **Planes:** A plane hypothesis can be generated using a single point (\mathbf{p}_0) with its normal direction ($\hat{\mathbf{n}}$). To test if a point \mathbf{p} lies on the plane $(\mathbf{p} - \mathbf{p}_0) \cdot \hat{\mathbf{n}} = 0$, the distance of the point from the plane $|(\mathbf{p} - \mathbf{p}_0) \cdot \hat{\mathbf{n}}|$ is used.
- **Cylinders:** A cylinder hypothesis can be generated using 2 points ($\mathbf{p}_0, \mathbf{p}_1$) with their normal directions ($\hat{\mathbf{n}}_0, \hat{\mathbf{n}}_1$). The principal axis of the cylinder is selected as the minimum distance line between the normal directions $\hat{\mathbf{n}}_0$ and $\hat{\mathbf{n}}_1$. The radius r is the distance of either point to this line. To test if a point \mathbf{p} with normal direction $\hat{\mathbf{n}}$ lies on the cylinder, the minimum distance point on cylinder's axis is calculated \mathbf{p}_{\min} . The vector $\mathbf{p} - \mathbf{p}_{\min}$ should have length r and direction $\hat{\mathbf{n}}$.
- **Spheres:** A sphere hypothesis can be generated using 2 points ($\mathbf{p}_0, \mathbf{p}_1$) with their normal directions ($\hat{\mathbf{n}}_0, \hat{\mathbf{n}}_1$). The intersection of the lines along the normal directions of each point is the center of the sphere \mathbf{p}_c . The radius r is the distance of either point to the center. To test if a point \mathbf{p} with normal direction $\hat{\mathbf{n}}$ lies on the sphere, vector having length r and direction $\mathbf{p} - \mathbf{p}_c$ is compared with the vector having length $|\mathbf{p} - \mathbf{p}_c|$ and direction $\hat{\mathbf{p}}$.

6.4.1.1 Energy function creation

The segmentation problem is posed as a multi-label graph-cuts optimization problem. A graph $G = \{V, E\}$ is constructed such that each point in the point cloud is a vertex $v_i \in V$. An edge E_{ij} connects

vertices v_i and v_j if the euclidean distance between the points represented by these vertices is less than a threshold.

$$E = \{E_{ij} | \| (v_i - v_j) \|_2 \leq r_{\max}\} \quad (6.1)$$

Labels $l_i \in L$ are defined such that each label represents a color. Each l_i is defined by a Gaussian $N(\mu_i, \Sigma_i)$ in the HSV space. Each of these vertices needs to be assigned a label which indicates the color of the object to which the point belongs. The energy term associated with this graph is defined by $D = D_p + D_s + D_l$

where,

$$D_p = \sum_{v_i \in V} \sum_{l_j \in L} -\log P(v_i, l_j) \quad (6.2)$$

$$P(v_i, l_j) = \exp(-(v_i - \mu_j) \times \Sigma_j^{-1} \times (v_i - \mu_j)^T) \quad (6.3)$$

is the data term. It represents the likelihood based only on the data contained in the node v_i that it belongs to the label L_j .

$$D_s = \sum_{v_i \in V} \sum_{v_j \in N_{v_i}} (k_1 \times \|(v_{i_xyz} - v_{j_xyz})\|^2 + k_2 \times \|(v_{i_hsv} - v_{j_hsv})\|^2 + k_3 \times \|(v_{i_n} - v_{j_n})\|^2) \quad (6.4)$$

where v_{i_hsv} , v_{i_xyz} , v_{i_n} represent the HSV, Cartesian co-ordinates and normal directions respectively for the point represented by the node v_i . N_{v_i} is the set of points lying in the neighborhood of the point represented by the node v_i . D_s is the smoothness term, which represents the energy due to spatially incoherent labels. It can be considered as an interaction term between neighboring nodes, where neighbors prefer to have same labels. k_1, k_2, k_3 are constants which influence the importance of each of the terms. For example, if $k_1 \gg k_2, k_3$, neighboring nodes having similar HSV values will prefer having similar labels, irrespective of their normal directions or euclidean distance from each other.

$$D_l = \sum_{l_i \in L} \sum_{l_j \in L} -\log P(l_i, l_j) \quad (6.5)$$

is the label swap term. It is an indication of the likelihood of swapping labels for a given vertex. $P(l_i, l_j)$ represents the probability that there is an ambiguity between labels l_i and l_j for a point. These terms are generally set offline using color models. In this context, the labels which are likely to get mixed up easily (e.g. white and metal) are assigned a higher probability whereas labels which are unlikely to get mixed up (e.g. red and blue) are assigned lower probability.

6.4.2 Primitive Shape assignment

The hypotheses associated with each point in the cloud can be considered as labels for point. There may be multiple labels associated with each point and the labeling may be spatially incoherent. To resolve such issues and generate a smooth labeling, a multi-label optimization using graph-cuts is performed. In this setting, the nodes in the graph comprise all possible assignment of labels to the points. The different terms of the energy functional are explained below:

- The data term indicating the cost of a label assignment to a point is proportional to the distance of the point from the primitive shape.

- The smoothness term penalizes neighboring points having different labels and the penalty is inversely proportional to the distance between the neighboring points.
- Label swap energies are used for neighboring primitive shapes in a way that only neighboring primitive shapes labels can be swapped.
- Label energies are added according to the complexity of the primitive shape (e.g. planes require 3 parameters while cylinders require 7), thereby favoring simpler primitives and penalizes over-fitting of primitives.

The convex energy functional thus created is then solved using the α - expansion, β -swap algorithms [99, 100, 101, 102] which gives the label assignment for each point in the cloud, such that the total energy is minimized.

6.4.3 Merging and pruning of Primitive Shapes

Due to the addition of label energies, some hypotheses might not be assigned any points. Such hypotheses are removed from the hypothesis set and the optimization process is repeated until no more hypotheses can be removed.

6.4.4 Primitive Shape Graph(PSG) representation

The primitive shapes detected in the previous step are now used to create a graphical representation of the point cloud. In this graph $G = (V, E)$, each primitive shape is a node $v \in V$ and neighboring primitive shapes are connected by an edge $e \in E$. An example of such a graph is shown in Fig. 6.3 (c).

6.5 Object Instance Detection

The object instance detection problem refers to the detection of all instances of a given model in a scene. We use information about primitive shapes in the scene to achieve this. A minimal set of primitive shapes (defined in Section 6.6.1) $\mathcal{P}_i \in \mathcal{P}_m$ in the model is chosen. A feature vector describing this set is calculated using Table 6.1. A successful match of this feature vector indicates a possible object instance.

6.5.1 Feature vectors for sets of Primitive Shapes

For detection of objects in the scene, correspondences between the scene shape primitives and model shape primitives need to be determined. We obtain these correspondences by computing and matching feature vectors from the geometric properties of the primitive shapes. These feature vectors not only encode the geometric properties of the shapes, but also of the relations between the shapes.

Some of the feature vectors used for primitive shapes and their combinations are defined in Table 6.1. The table only defines some simple combinations of feature vectors. However, more complicated features involving more primitive shapes can be constructed from these representations by simple extensions.

Using this approach, a feature vector can be constructed for the entire object as well. The visibility of primitive shapes depends on the viewpoint of the camera. Hence, feature vectors for all possible sets of primitive shapes should be calculated offline for the object models. Minimal sets of primitives from

Table 6.1: Feature vectors for primitive shape sets

Primitive shape	Feature Vector (FV)
Inf. Plane (Pl)	ϕ
Sphere (Sp)	radius
Inf. Cylinder (Cyl)	radius
Pl ₁ +Pl ₂	FV(Pl ₁), FV(Pl ₂), $\angle(\mathbf{n}_1, \mathbf{n}_2)$, min_distance(Pl ₁ , Pl ₂)
Pl+Cyl	FV(Cyl), FV(Pl), $\angle(\mathbf{n}, \mathbf{a})$
Cyl ₁ +Cyl ₂	FV(Cyl ₁), FV(Cyl ₂), $\angle(\mathbf{a}_1, \mathbf{a}_2)$, min_distance(Cyl ₁ , Cyl ₂)
Pl ₁ +Pl ₂ +Cyl	FV(Pl ₁ , Cyl), FV(Pl ₂ , Cyl)

Table 6.2: Summary of supported constraints between primitive shapes

Constraint (i)	Cost Function (g_i)	Bounds (lb, ub)	Constrained Spaces
Pl ₁ -Pl ₂	$[\mathbf{n}_1^T \hat{\mathbf{p}}_{21}; \hat{\mathbf{n}}_2^T \mathbf{n}_1]$	lb : $[d_{\min}; a_{\min}]$ ub : $[d_{\max}; a_{\max}]$	\mathbf{C}_n : $[\mathbf{n}_{1_{\perp 1}}; \mathbf{n}_{1_{\perp 2}}]$ \mathbf{C}_p : $[\mathbf{n}_1]$
Cyl ₁ -Cyl ₂	$[\ \hat{\mathbf{p}}_{21} - (\mathbf{n}_1^T \hat{\mathbf{p}}_{21})\mathbf{n}_1\ _2^2; \hat{\mathbf{n}}_2^T \mathbf{n}_1]$	lb : $[d_{\min}^2; a_{\min}]$ ub : $[d_{\max}^2; a_{\max}]$	\mathbf{C}_n : $[\mathbf{n}_{1_{\perp 1}}; \mathbf{n}_{1_{\perp 2}}]$ \mathbf{C}_p : $[\mathbf{n}_1]$
Sph ₁ -Sph ₂	$[\hat{\mathbf{p}}_{21}]$	lb : d_{\min} ub : d_{\max}	\mathbf{C}_n : $[\mathbf{n}_{1_{\perp 1}}; \mathbf{n}_{1_{\perp 2}}]$ \mathbf{C}_p : $[\mathbf{n}_1]$

the scene point cloud are calculated during the pose estimation stage (Section 6.6.3), and the distance between the feature vectors provides a metric for obtaining hypotheses of shape associations.

6.6 Object Pose Estimation

After hypotheses of object instances in the scene are determined, the corresponding poses are estimated using a constraint-based approach. This section describes the steps involved in determining the object pose from a set of primitive shape matches between the model and scene.

6.6.1 Geometric constraints from primitive shape matching

Each primitive shape \mathcal{P}_i enforces a set of constraints ($\mathbf{C}_{pi}, \mathbf{C}_{ni}$) on the position and orientation of the object respectively. Each row of \mathbf{C}_{pi} and \mathbf{C}_{ni} contains a direction along which the constraint has been set. Examples of constraints set by each primitive shape are explained below:

- Infinite plane: $\mathbf{C}_{pi} = [0, 0, 1]$ and $\mathbf{C}_{ni} = [1, 0, 0; 0, 1, 0]$
- Infinite cylinder: $\mathbf{C}_{pi} = [1, 0, 0; 0, 1, 0]$ and $\mathbf{C}_{ni} = [1, 0, 0; 0, 1, 0]$
- Sphere: $\mathbf{C}_{pi} = [1, 0, 0; 0, 1, 0; 0, 0, 1]$ and $\mathbf{C}_{ni} = \phi$

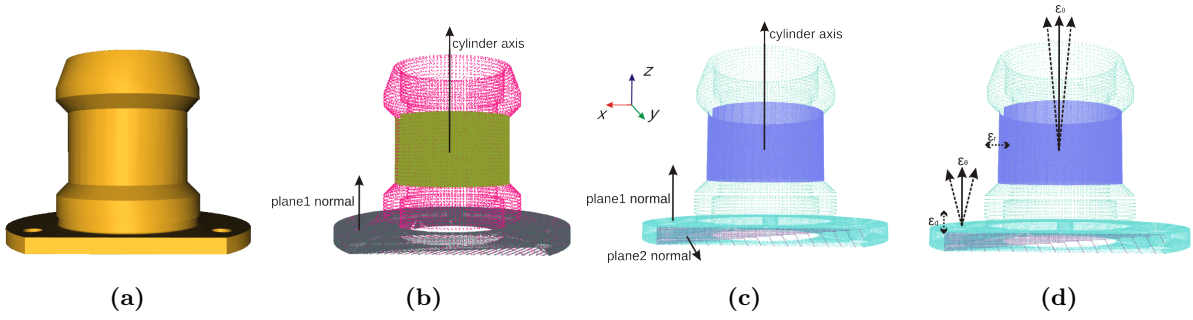


Figure 6.4: Primitive shape groups in an industrial workpiece. (a) Object model. (b) The position of the object is fully defined in 3D. However, rotation of the object along the z -axis is not fixed. (c) Position and orientation of the object are both fully defined and these primitive shapes form a minimal set. (d) The expected errors or tolerances in estimation of primitive shapes from point cloud data are shown (exaggerated).

A *complete set* of primitive shapes is defined as a set where the constraints fully specify the 3D position and orientation of the object. A *minimal set* of primitive shapes is defined as a set which is complete but removing any primitive shape from the set would render it incomplete.

Table 6.2 presents the list of supported geometric constraints between primitive shapes, where

$$\begin{aligned}\hat{p}_2 &= \mathbf{R}p_2 + t \\ \hat{p}_{21} &= \hat{p}_2 - p_1 \\ \hat{n}_2 &= \mathbf{R}n_2\end{aligned}\tag{6.6}$$

6.6.2 Detection of minimal and complete sets of primitives

The constraints (C_{pi}, C_{ni}) enforced by each primitive shape \mathcal{P}_i are stacked into two matrices C_p and C_n (each having 3 columns), where C_p represents the combination of constraints for the position and C_n represents the combination of constraints on the orientation. The constraints are complete if the matrices C_p and C_n both have rank 3. Fig. 6.4 shows an example of a complete set of primitive shapes. An algorithm for detecting minimal sets is presented in Algorithm 9.

6.6.3 Constraint solving for pose estimation

Let us consider the object's frame to be specified at position \mathbf{t} and orientation (rotation matrix) \mathbf{R} . This can also be represented in the form of a homogeneous transformation matrix \mathbf{T} . From the object models, the relative pose (partially defined) \mathbf{T}_i of each primitive shape \mathcal{P}_i w.r.t. the object's frame is available.

The optimization is performed over transformations that align the model to the objects in the scene. The transformations are represented as $\Delta\mathbf{x} = (\mathbf{t}, \mathbf{r})$ where \mathbf{t} is the translation and \mathbf{r} is the rotation in axis angle representation.

6.6.3.1 Iterative solver

The function to be optimized is the absolute value of the transformation. In other words, this means minimizing $\|\Delta\mathbf{x}\|_2$. The constraint functions g_i along with their limits $(lb(g_i), ub(g_i))$ are obtained from

```

Input:  $\mathcal{P}$  (set of primitive shapes)
Output:  $\mathcal{P}_{\min}, \mathcal{P}_{\text{complete}}$  (sets of minimal and complete primitive shapes)
foreach  $\mathcal{P}_i \in \mathcal{P}$  do
  minFlag  $\leftarrow$  true
   $\mathcal{P}_{\text{candidate}} \leftarrow \mathcal{P}_i$ 
  foreach  $\mathcal{P}_j, j > i$  do
     $\mathcal{P}_{\text{candidate}} \leftarrow \mathcal{P}_{\text{candidate}} \cup \mathcal{P}_j$ 
    if CheckComplete( $\mathcal{P}_{\text{candidate}}$ ) then
       $[\mathcal{P}_i]_{\text{complete}} \leftarrow [\mathcal{P}_i]_{\text{complete}} \cup \mathcal{P}_{\text{candidate}}$ 
      if minFlag then
        minFlag  $\leftarrow$  false
         $[\mathcal{P}_i]_{\min} \leftarrow [\mathcal{P}_i]_{\min} \cup \mathcal{P}_{\text{candidate}}$ 
      end
    end
  end
end

```

Algorithm 9: Detecting minimal and complete primitive shape sets

the primitive shape matching constraints shown in Table 6.2. The (d_{\min}, d_{\max}) values of the constraints can be used to incorporate the noise in sensor data or primitive shape fitting errors (Section 6.8.1), as well as manufacturing uncertainties (Section 6.8.5). As an example shown in Fig. 6.4: If the error in estimation of a cylinder's radius (r) is ϵ_r , the bounds for the cylinder matching constraint can be set as $(d_{\min}, d_{\max}) = (r - \epsilon_r/2, r + \epsilon_r/2)$. For an error ϵ_θ in estimation of the normal direction, the bounds can be set as $(a_{\min}, a_{\max}) = [\cos(\epsilon_\theta), 1]$.

The resulting optimization problem is:

$$\begin{aligned}
 & \arg \min_{\Delta \mathbf{x}} \quad \|\Delta \mathbf{x}\|_2 \\
 & \text{subject to} \quad lb(g_i) \leq g_i \leq ub(g_i), \quad i = 1, \dots, m.
 \end{aligned} \tag{6.7}$$

This set of equations is then solved using a non-linear least squares min-max solver (SLSQP) from the optimization framework NLOpt.

In this way, the constraints defined by the primitive shapes can be solved to obtain a pose of the object. If the constraints are complete, the pose is uniquely defined. Otherwise, the constraint solver returns one possible solution.

One advantage of using the iterative solver is that in the presence of noisy data, it provides a best fit solution in the least-squares sense. Noisy data and uncertainties in primitive shape estimation are common in computer vision. Hence, this solver is good at finding the optimal pose that tries to minimize these errors.

Input: $[\mathcal{P}_s, [\mathcal{P}_m]_{\min}]$ (set of scene primitive shapes and minimal sets of model primitive shapes)
Output: $[T, h_{\max}]$ (best pose estimate with score for detected object instance)
foreach $\mathcal{P}_i \in [\mathcal{P}_m]_{\min}$ **do**
 $h_{\max} \leftarrow 0$
 compute shape matching hypothesis (H_i) using fv's, Section 6.5.1
 calculate transformation estimate T_i for H_i , Section 6.6.3
 compute score h_i for hypothesis H_i
 if $h_i \geq \text{thresh} \ \&\& \ h_i > h_{\max}$ **then**
 $T \leftarrow T_i$
 $h_{\max} \leftarrow h_i$
 end
end

Algorithm 10: Detecting object poses using RANSAC

6.6.3.2 Exact solver

The constraints C_i that restrict the object's pose can also be solved using the exact geometric solver in Section 3.4. The transformation manifolds obtained from the constraint solver represents the underspecification of the robot's pose.

The runtime for generating each hypothesis is an important concern in the object pose estimation problem. This is especially critical in case of objects with a large number of primitive shapes, since the number of hypotheses can be large. The main advantage of using the exact solver for this application is its runtime. The runtime for estimating the pose per hypothesis is deterministic and also significantly faster than the iterative solver.

6.6.4 RANSAC based constraint solving for pose estimation

A shape matching hypothesis H_i consists of a set of associations between primitive shape sets, i.e., correspondences that relate each primitive shape in the scene to a primitive shape in the model. The hypotheses can be computed by matching feature vectors, Section 6.5.1. An algorithm for pose estimation using RANSAC-like iterations on minimal sets of primitive shapes is described in Algorithm 10.

We use an efficient hypothesis verification approach that utilizes the geometric information from CAD models and primitive shape decomposition of scene point clouds, as described in [90].

6.7 Evaluation

There are two important modules in this work: primitive shape decomposition, and object detection and pose estimation. We evaluate both these modules and present the results in the following sections.

6.7.1 Shape-based segmentation of point clouds

The Object Segmentation Database [103] was used to evaluate this algorithm. Fig. 6.5 shows the results from primitive shape decomposition of scene clouds taken from this database. The focus here is to

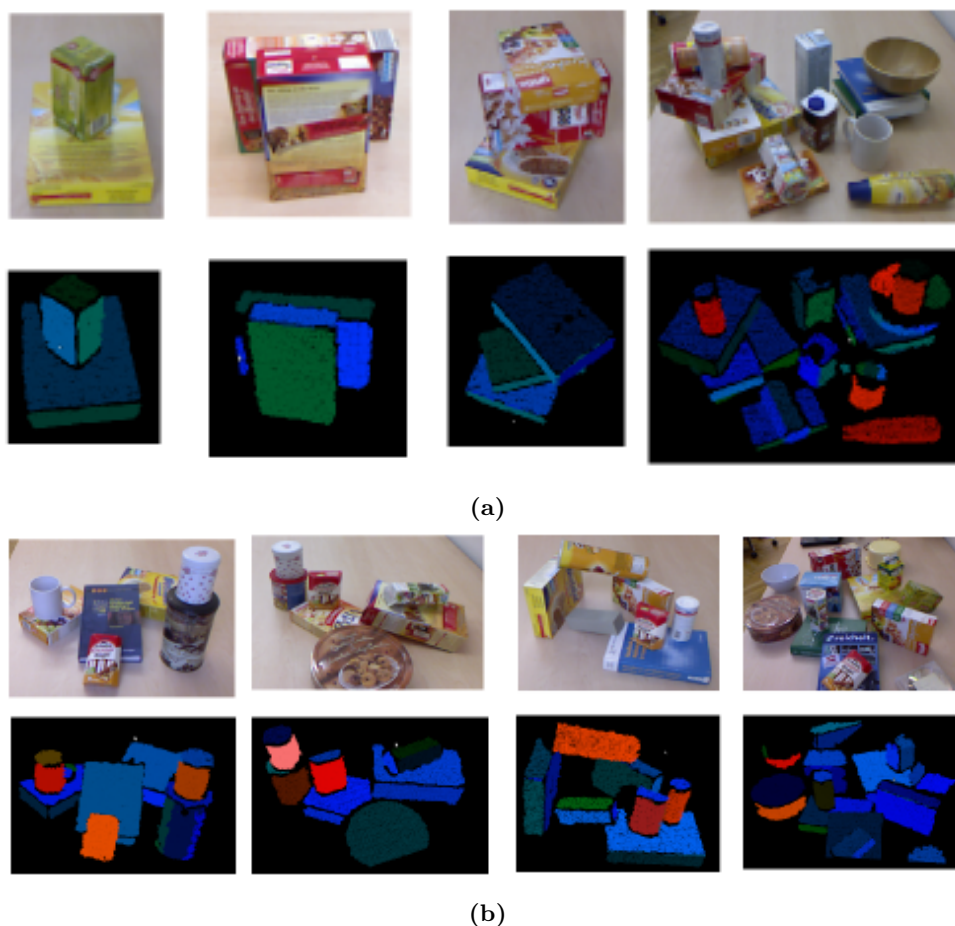


Figure 6.5: Primitive Shape Detection results. Cylinders are shown in red and planes are shown in blue-green.

show that our approach works on a wide variety of objects, as seen in this dataset. We don't perform quantitative evaluations since this module is essential for object detection and pose estimation but not claimed as a major contribution.

6.7.2 Constraint-based object detection and pose estimation

We conduct several experiments in simulation to analyze the performance of our algorithm. We choose simulated experiments so that perfect ground truth data is available to ensure correctness of the object pose estimates. Also, the focus in this work is on the qualitative aspects of detecting underspecified object poses, and the runtime rather than errors in pose estimation. The errors in pose estimation arise largely from the errors in primitive shape detection, since the exact solver's accuracy is only limited by numerical limitations.

We evaluate the pose estimation runtime on several objects in our dataset. Each object is decomposed into a set of primitive shapes. Fig. 6.6 shows the 3D models of these objects and their primitive shape decompositions. We also mention the number of primitive shapes detected in the actual scene, since this directly affects the number of pose hypotheses. The results from this experiment are summarized in

6. CONSTRAINT-BASED OBJECT DETECTION AND POSE ESTIMATION

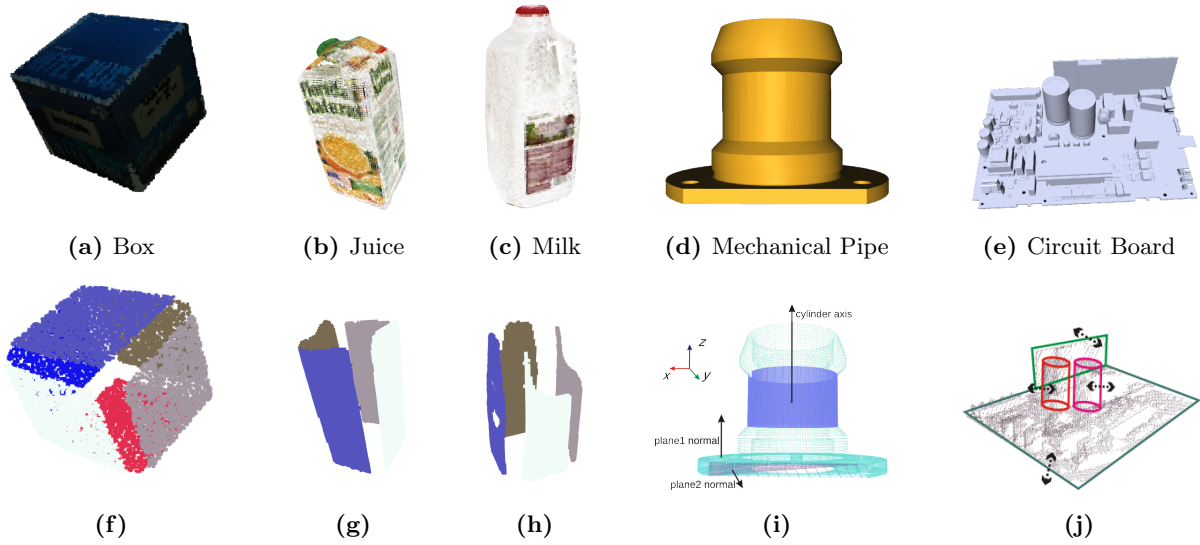


Figure 6.6: RGBD Object models (a,b,c,d,e) and corresponding primitive shape decompositions (f,g,h,i,j)

Table 6.3: Evaluation of object pose estimation

Model	#PS Model	#PS Scene	#Hypotheses	Iterative [7] Runtime (in ms)	Exact. [6] Runtime (in ms)
Box	6	6	64	62 ± 2	3.4 ± 0.01
Juice	4	4	-	0.0 ± 0.0	0.00 ± 0.00
Milk	4	4	-	0.0 ± 0.0	0.00 ± 0.00
Mech. Pipe	3	3	1	0.0 ± 0.0	0.00 ± 0.00
Circuit Board	4	4	20	N/A	1.0 ± 0.01

Table 6.3. It is clear from the table that the exact solver is significantly faster than the iterative solver, and hence much more suitable for this application. This difference becomes more evident with increasing number of pose hypotheses.

The time required for hypothesis verification depends directly on the number of hypotheses. The number of hypotheses depends on the number of complete and minimal cliques, which in turn depends on the number of primitive shapes. Hence, we evaluate the number of minimal and complete cliques for different objects. The results are summarized in Table 6.4.

6.8 Applications

There are several applications and scenarios where our constraint-based object detection and pose estimation approach proves useful and provides a clear advantage w.r.t state-of-art approaches. Some such examples are listed in the following sections.

Table 6.4: Minimal and complete sets for object pose estimation

Model	#Primitive shapes	#Complete cliques	#Minimal cliques
Box	6	27	8
Juice	4	-	-
Milk	4	-	-
Mech. Pipe	3	1	1
Circuit Board	4	3	2

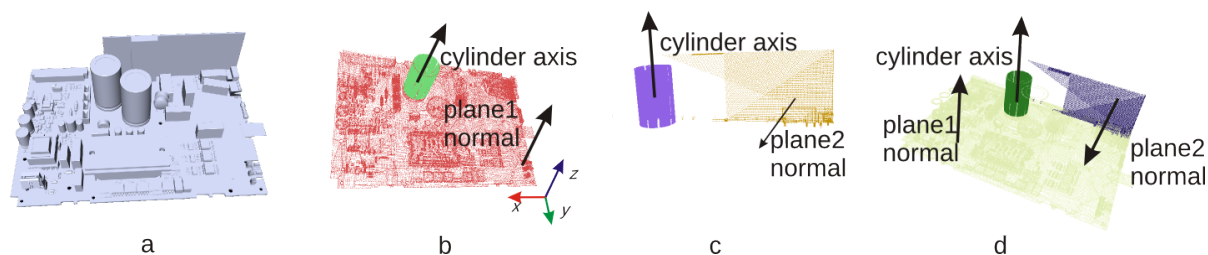


Figure 6.7: Primitive shape groups. (a). Object model. The cylinder fixes the position of the object in x and y axes, Pl_1 fixes the position along z-axis, and Pl_2 fixes the position in y-axis. The cylinder and Pl_1 both fix rotation of the object in x and y axes, and Pl_2 fixes rotation in z and x axes. (b) Position of the object is fully defined in 3D. However, the cylinder axis direction and normal direction of Pl_1 are the same. Hence, rotation of the object along z-axis is not fixed. (c) Position and orientation of the object are fixed and these primitive shapes form a minimal set. (d) Position and orientation of the object are fully defined. These primitive shapes form a complete set but not a minimal set.

6.8.1 Calculating object poses from noisy sensor data

The primary application of this work is the detection and pose estimation of objects from noisy sensor data. Primitive shapes such as planes, cylinders and spheres are detected from RGBD data, as explained in [90]. This step also provides estimates of the shape fitting errors. These noise estimates can be expressed as inequality constraints and solved using the constraint solving algorithms. This pipeline is explained in Fig. 6.1. Fig. 6.7 shows examples of (b) not complete, (c) minimal and (d) complete sets of primitive shapes for the same object.

6.8.2 Calculating detectability of objects from viewpoints

Depending on the viewpoint of the camera, different sets of primitive shapes might be visible in the scene point cloud. Also, the accuracy of the primitive shape detection depends on the distance of the object and the viewpoint, as shown in Fig. 6.8. Using Algorithm 9, it can be ascertained whether the object is detectable from a given viewpoint or not. This knowledge is important in object recognition and pose estimation problems, where primitive shape based approaches might fail due to insufficient data available from a certain viewpoint.

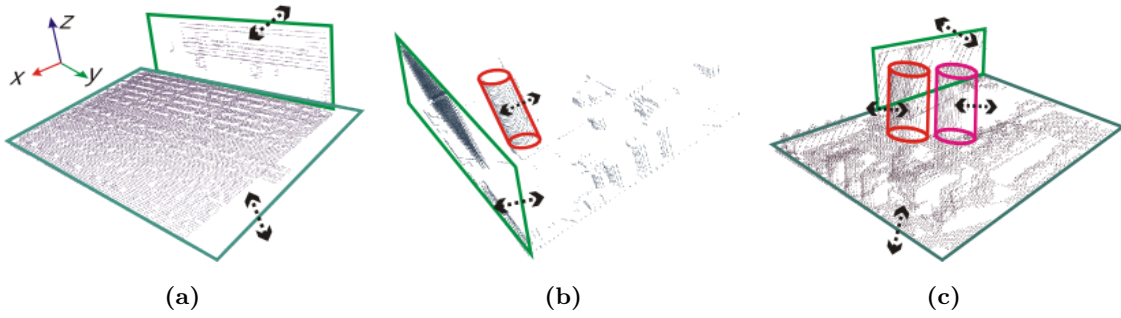


Figure 6.8: Primitive shape groups for different views of an object. Using primitive shape sets, it can be calculated whether an object’s pose can be fully estimated from a viewpoint or not. The arrows indicate the expected noise (qualitative only) in estimation of the primitive shape parameters. (a) the position of the object along y axis is not determined. In (b) and (c) the complete pose of the object can be estimated. (Note: The detected primitive shapes are highlighted manually to make the image clearer.)

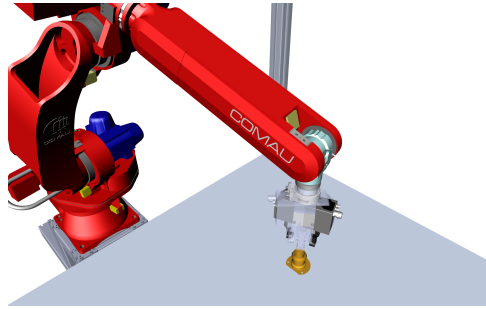


Figure 6.9: Grasping of a symmetric object: The grasp pose for the object is invariant to rotations along the symmetrical axis of the cylindrical object. Two such grasping poses are shown in this figure.

6.8.3 Reasoning about symmetrical objects

The ontological representation of CAD models is used to obtain geometric information about the CAD models. The information about minimal and complete sets of primitive shapes can be propagated back to the ontology and utilized by different applications. Information about the nullspace of geometric constraints, now available in the ontology, can also be used for several robotic manipulation applications, Section 6.8.4.

6.8.4 Manipulation of symmetric objects

Knowledge about the symmetrical properties of objects/workpieces can be used to optimize robotic manipulation tasks involving these objects. Fig. 6.9 shows a grasping task involving a symmetrical object. Using the object recognition approach presented in this thesis, the robotic system is aware of the fact that the z-axis of the object is an axis of symmetry. Hence, as illustrated in Fig. 6.9, the object can be grasped using any orientation of the end effector along the z-axis. This allows optimization of the robot’s pose within this nullspace. More such applications have been described in our works [2, 65].

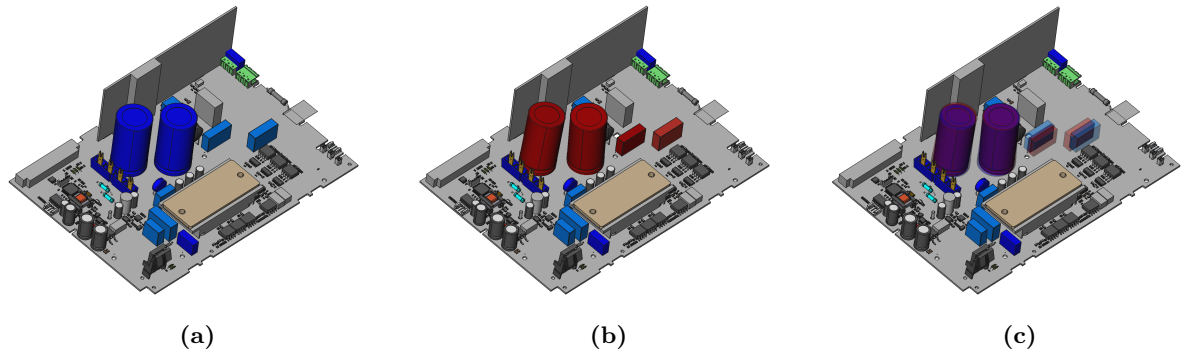


Figure 6.10: Manufacturing tolerances in production or assembly of parts. (a) Nominal object CAD model (b) Actual manufactured model, with parts in Red having some placement errors (c) Superimposed image of the perfect and manufactured object, that illustrates the manufacturing tolerances.

6.8.5 Manufacturing uncertainties in object models

The manufactured objects might differ from the nominal CAD models due to manufacturing tolerances. In case of assembled objects, there can also be errors in the placement of individual parts. Fig. 6.10 shows an electronic circuit, where the capacitors (blue cylinders) are supposed to be mounted perpendicular to the circuit board. On an actual workpiece, the capacitors (red cylinders) are mounted at a slightly different angle. Also, the resistors (blue cuboids) are mounted at a slightly different location on the real workpiece (red cuboids). These are not errors but allowed tolerances in the manufacturing process. Using our object recognition approach (Sections 6.6.1 and 6.6.3), these uncertainties can be modeled explicitly and even such imperfect object models can be detected accurately.

6.9 Discussion

In this Chapter, we presented an approach for object detection and pose estimation using geometric constraints. This approach is especially suitable for texture-less parts which are composed of a small number of simple geometries such as planes and cylinders. We show how this constraint-based approach can be used to account for estimated noise in the sensor data, manufacturing tolerances, and a systematic handling of symmetrical objects.

In the next Chapter, we present a constraint-based object tracking algorithm. The geometric constraints from primitive shape matching and partial poses estimated from the algorithms in this Chapter work as initializations for the tracker in the next Chapter.

Chapter 7

Tracking using Primitive Shape Constraints (TPSC)

Visual tracking of an object in 3D using its geometrical model is an unsolved classical problem in computer vision. The use of point cloud (RGBD) data for likelihood estimation in the state estimation loop provides improved matching as compared to 2D features. However, point cloud processing is computationally expensive given its big data nature, making the use of mature tracking techniques such as Particle Filters challenging. For practical applications, the filter requires implementation on hardware acceleration platforms such as GPUs or FPGAs.

In this chapter, we introduce a novel approach for object tracking using an adaptive Particle Filter operating on a point cloud based likelihood model. The novelty of the work comes from a geometric constraint detection and solving system which helps reduce the search subspace of the Particle Filter. At every time step, it detects geometric shape constraints and associates it with the object being tracked. Using this information, it defines a new lower-dimensional search subspace for the state that lies in the nullspace of these constraints. It also generates a new set of parameters for the dynamic model of the filter, its particle count and the weights for multi-modal fusion in the likelihood modal. As a consequence, it improves the convergence robustness of the filter while reducing its computational complexity in the form of a reduced particle set.

7.1 Introduction

Model based visual tracking has been and continues to be an extensively researched area within computer vision. It finds application in several domains within robotics and automation. The problem has been explored both in *2D* and *3D* w.r.t. the sensor used and the dimension of the estimated state space. It continues to be a challenging problem given the nonlinearity and noise seen in the available vision sensors. A comprehensive survey of object tracking methods and systems mainly using RGB cameras has been covered in [104]. The recent availability of low cots RGBD sensors have made it possible to explore model based tracking using point cloud data.

In visual tracking, typically the state space approach is adopted to model the position, velocity and acceleration of the object to be tracked. Statistical correspondence methods are used to estimate the

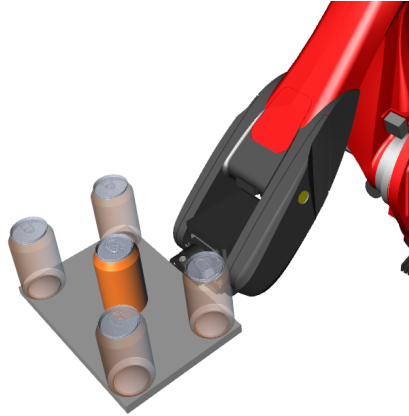


Figure 7.1: Constrained motion of a manipulated object: the soda can lies on a tray that is being moved by a robot.

object's state by collecting sensor measurements, where model uncertainties and sensor noise is also modeled. The Kalman Filter and Particle Filter are the two most popular methods in model based tracking [105]. The Kalman filter assumes the state space to be distributed by a Gaussian and that the update model has a linear relation with the measurement model. If the state update and measurement models are non linear, the Extended Kalman filter [106] is used where a linear approximation of the same are obtained using the Taylor series expansion. The assumption of a Gaussian state space distribution limits the application of a Kalman filter. The Particle Filter overcomes this limitation by representing the conditional state density using a set of samples called particles, where each particle contains a weight obtained from the measurement (likelihood). Particle Filters are observed to be robust under multi-modal likelihoods due to clutter in the tracking scene and thereby are much better in avoiding local minima as compared to Kalman filters [107], [108]. However, the computation cost associated with Particle Filters increases exponentially with the dimensionality of the modeled state space and the complexity of the measurement model. For tracking using point cloud data, the measurement models are inherently computation and memory intensive. Therefore, known applications of Particle Filter for point cloud based object tracking is rather limited [109]. Nevertheless, particles filter operating on point cloud data and color (RGBD) information serve as a powerful tool for model based object tracking.

For our approach, an important assumption that we make about object models is that some parts of their geometries can be approximated using a set of primitive shapes such as planes, cylinders, spheres, etc. For object recognition and pose estimation, these primitives prove to be very useful since they can be detected more accurately and efficiently than complete CAD models.

Whenever an object interacts with its environment or a manipulator, its motion and dynamics are affected. With information about the primitive shapes in the environment, these restrictions can be modeled as geometric constraints between primitive shapes (Fig. 7.1).

Effective exploration and exploitation of primitive shape constraints can lead to considerable reduction in the computational complexity of Particle Filters used for model based object tracking using RGBD sensors. These constraints originate from the following sources:

- geometric properties intrinsic to the object model

- constraints imposed by the environment on the object on contact and interaction
- constraints imposed by a manipulator, influencing the dynamics of the object
- partial views based on sensor viewpoint

While the object interacts with the environment, real-time detection of the above mentioned primitive shape constraints can enable simplification of the object dynamics and thereby reduce the state space dimensions of the Particle Filter’s sample set. Furthermore, the number of particles required to estimate the state can be reduced given the dimension reductions. As a consequence the computational effort required by the Particle Filter can be considerable reduced. We demonstrate a novel method for achieving the above mentioned concepts and a systematic implementation with evaluations in simulation and a real-world setup. The methodology demonstrates the feasibility of using Particle Filters with RGBD sensors in real world applications.

The rest of the chapter is organized as follows: Section 7.2 presents the state of the art in model based object tracking followed by Section 7.3 which presents the novel concept of tracking with primitive constraints. In Section 7.4 the adaptive Particle Filter using primitive shape constraints is introduced. Finally, experiments and evaluations are presented in Section 7.5 followed by conclusion and future work in Section 7.6.

7.2 Related Work

Object recognition and pose estimation of objects from 3D data is a classical problem in computer vision. Here we refer to a memoryless, single-frame system that can be used to initialize or correct the tracker. In our previous work [8, 9], we have presented approaches for estimating the partial pose of objects and also the remaining DoFs. Other work in this area features primitive shape graphs [90, 94], surflet-pair based approaches [90, 92], and the triple-point feature method [93]. Keypoint and descriptor based approaches include local shape keypoint [81] and global descriptors [88].

Model based object tracking in general finds several citing in computer vision literature. Several algorithms and systems have been developed over the years. The work presented in [110] provides a unifying framework for model based visual tracking in the form of a library of algorithms for 2D and 3D tracking, where object tracking using a variety of sensors and visual features can be explored. Similarly, [111] developed a framework for automatic modeling, detection, and tracking of 3D objects using RGBD data from sensors such as a Kinect. They employ a multi-modal template based approach presented in [112]. In [113] the authors present a GPU based framework for accelerating Particle Filter based approaches applied to 3D model-based visual tracking.

A special case of 3D visual tracking is tracking of human hands in several degrees of freedom (3-26). Multiple approaches in terms of visual features, tracking methods and data fusion have been explored for handling this high dimensional problem [114], [115], [116] and [117].

In terms of Particle Filter based tracking methods, several approaches have been documented in literature [118]. The work in [119] presents a Bayesian framework for 3D tracking of deformable objects by exploring multiple-cues. The authors propose a spatio-temporal model of the object using Dynamic Point Distribution Models (DPDMs) in order to improve robustness towards appearance changes. The

7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)

Input: Primitive Shape model $\mathcal{P}_{\text{model}}$, last object pose s_{t-1} , point cloud I_t , particle set \mathcal{S}_{t-1} , default Gaussian noise covariance \mathbf{w}^{def}

Output: Tracked object pose s_t , particle set \mathcal{S}_t

$\mathcal{P}_{\text{scene}} \leftarrow \text{detectPrimitives}(I_t)$

$\mathcal{C} \leftarrow \text{detectConstraints}(\mathcal{P}_{\text{model}}, \mathcal{P}_{\text{scene}}, s_{t-1})$

$\mathcal{C}_{\text{comb}} \leftarrow \text{combineConstraints}(\mathcal{C})$

foreach $\mathcal{S}^i = (s^i, \mathbf{w}^i)_{t-1} \in \mathcal{S}_{t-1}$ **do**

$\hat{s}_t^i \leftarrow \text{f}_{\text{proj}}(s_{t-1}^i)$ (Table 7.1)

$\hat{\mathbf{w}}_t^i \leftarrow [(\mathbf{N}_t \cdot \mathbf{w}_t^{\text{def}})\mathbf{N}_t + (\mathbf{S} \cdot \boldsymbol{\sigma}_t)\mathcal{S}_t, (\mathbf{N}_R \cdot \mathbf{w}_R^{\text{def}})\mathbf{N}_R + (\mathbf{S}_R \cdot \boldsymbol{\sigma}_R)\mathcal{S}_R]$ (Table 7.1)

end

$\mathcal{L} \leftarrow \text{computeLikelihood}(\hat{\mathcal{S}}_t)$

$\mathcal{S}_t \leftarrow \text{resample}(\hat{\mathcal{S}}_t, \mathcal{L})$

$s_t \leftarrow \text{mean}(\mathcal{S}_t)$

Algorithm 11: Algorithm for tracking with primitive shape constraints (TPSC)

state estimation is performed using a Particle Filter using three cues (shape, texture and depth) in the likelihood model. In terms of adaptive particle filters, the KLD tracker [120] is an approach for adapting the sample set based on the covariance values. In our approach, we adapt both the sample size and the tracking subspace based on the primitive shape matching constraints.

Many of the above tracking papers above were also single camera approaches, allowing nearest neighbor operations to be fast by exploiting the organized nature of 3D data [113]. To handle complete occlusion of objects multiple sensors would be required. Multi-camera tracking approaches were also studied [121] [122], but these approaches were found to either use individual RGB cameras directly during likelihood estimation, or extracted primitives directly from a fused point cloud scene for comparison.

Our core contribution is a novel approach for object tracking using multiple RGBD sensors through an adaptive Particle Filter. The adaptive nature of the filter is modeled on the use of a primitive shape detection algorithm which exploits the detected geometric constraints connected to the object during the tracking process and thereby exploits these constraints to reduce the search subspace of the Particle Filter increasing its convergence robustness and also reduces its computational complexity. Our approach can also use multiple sensors seamlessly, and works on unorganized point clouds via our fast GPU implementation of the following indexed octree library [123].

7.3 Primitive Shape Constraints for Tracking

This section describes our adaptive tracking approach with primitive shape constraints. The overall steps are described in Algorithm 11, and their individual details are described in the following subsections.

7.3.1 Object models based on primitive shapes

We use a boundary representation (BREP) [2] for all geometric entities in our system. This representation composes objects from semantically meaningful primitive shapes (e.g., planes, cylinders). Fig. 7.2 illustrates such Primitive Shape Decompositions for some of the objects used in this work. This object

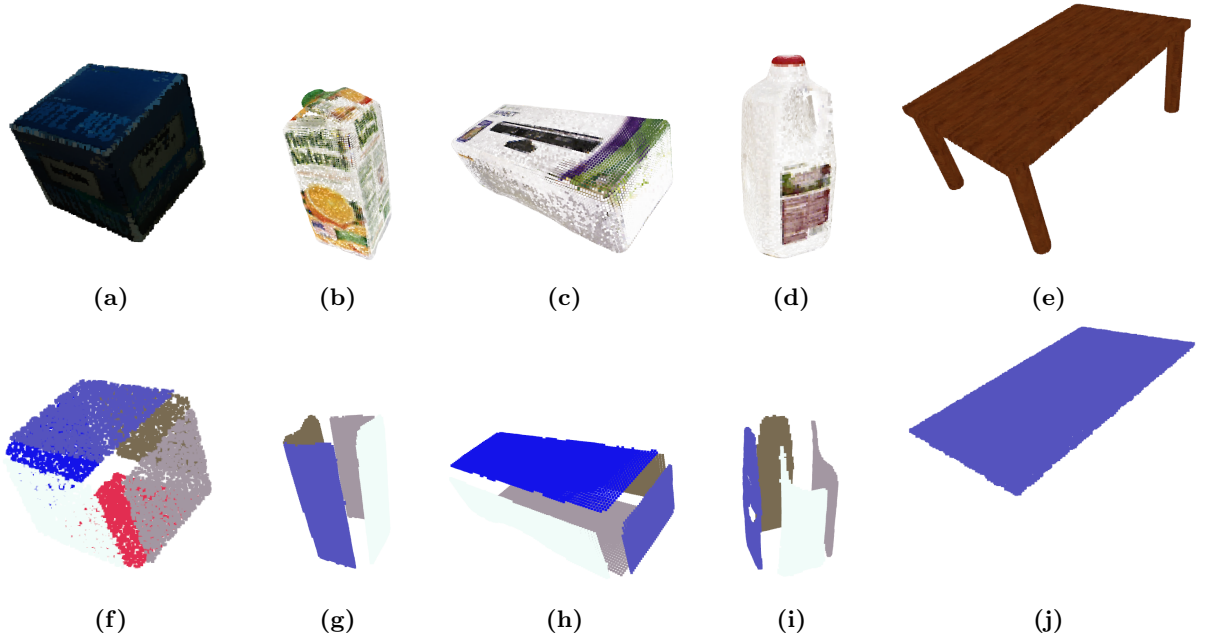


Figure 7.2: RGBD Object models (a,b,c,d,e) and corresponding primitive shape decompositions (f,g,h,i,j)

model is motivated from the observation that the estimation of primitive shapes such as planes and cylinders in a point cloud is faster and more robust than detection of the full object models. Based on this simplified model, the object detection and pose estimation problem can be considered as a primitive shape matching problem [9] between the set of model and scene primitive shapes (\mathcal{P}_m and \mathcal{P}_s respectively). In the context of object pose estimation, the model primitive can be considered as a constrained shape, the scene primitive as a fixed shape, and the shape match as a constraint $C_i = \text{Coincident}(\mathcal{P}_s^i, \mathcal{P}_m^i)$. A geometric constraint adds restrictions to the relative pose of the constrained shape w.r.t. the fixed shape.

Each primitive shape $\mathcal{P}_i \in \mathcal{P}$ enforces a set of constraints $C_i = (\mathbf{C}_{p_i}, \mathbf{C}_{n_i})$ on the position and orientation of the object respectively. Each row of \mathbf{C}_{p_i} and \mathbf{C}_{n_i} contains a direction along which the constraint has been set. Table 7.1 defines these nullspaces for the geometric constraints relevant to this work.

By analyzing the set of primitive shapes that form an object, properties such as symmetry can be determined. Each primitive shape $\mathcal{P}_i \in \mathcal{P}$ in the object is used to create a constraint $C_i = \text{Coincident}(\mathcal{P}_i, \mathcal{P}_i)$. The set of constraints $\mathcal{C} = \bigcup_i C_i$ is then used by a constraint solver (Chapter 3) to determine a transformation manifold. These transformation manifolds are geometric entities, and provide projection functions (Table 7.2). Hence, any pose hypothesis for the object can be projected onto this transformation manifold. The DoFs of this transformation manifold determine the symmetry axes of the object. This information about the symmetry axes and the projection functions from the transformation manifold can be exploited to improve the tracking, as shown in Section 7.4.

Given a partial view of an object, it might not be possible for a detector to determine its complete pose. This is especially the case for detectors based on primitive shapes, where fine features that form a small fraction of the object are often ignored in simplified models. Given the set of primitive shapes visible from a specific viewpoint, the DoF analysis can be used to determine the axes along which the pose can't be determined. This information can be utilized by the tracker to sample more particles in

7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)

Table 7.1: Nullspace definitions for supported geometric constraints

Fixed	Constrained	Constraint (C)	Controlled Space (S)	Null Space (N)
$\text{Pl}_1(\mathbf{p}_1, \hat{\mathbf{n}}_1)$	$\text{Pl}_2(\mathbf{p}_2, \hat{\mathbf{n}}_2)$	Dist. (d)	$S_R : [\hat{\mathbf{n}}_{1\perp 1}; \hat{\mathbf{n}}_{1\perp 2}]$ $S_t : [\hat{\mathbf{n}}_1]$	$N_R : [\hat{\mathbf{n}}_1]$ $N_t : [\hat{\mathbf{n}}_{1\perp 1}; \hat{\mathbf{n}}_{1\perp 2}]$
$\text{Pl}_1(\mathbf{p}_1, \hat{\mathbf{n}}_1)$	$\text{Pl}_2(\mathbf{p}_2, \hat{\mathbf{n}}_2)$	Coinc.	$S_R : [\hat{\mathbf{n}}_{1\perp 1}; \hat{\mathbf{n}}_{1\perp 2}]$ $S_t : []$	$N_R : [\hat{\mathbf{n}}_1]$ $N_t : [1]$
$\text{L}_1(\mathbf{p}_1, \hat{\mathbf{a}}_1)$	$\text{L}_2(\mathbf{p}_2, \hat{\mathbf{a}}_2)$	Coinc.	$S_R : [\hat{\mathbf{a}}_{1\perp 1}; \hat{\mathbf{a}}_{1\perp 2}]$ $S_t : [\hat{\mathbf{a}}_{1\perp 1}; \hat{\mathbf{a}}_{1\perp 2}]$	$N_R : [\hat{\mathbf{a}}_1]$ $N_t : [\hat{\mathbf{a}}_1]$
$\text{Pt}_1(\mathbf{p}_1)$	$\text{Pt}_2(\mathbf{p}_2)$	Coinc.	$S_R : []$ $S_t : 1$	$N_R : [1]$ $N_t : []$
$\text{L}_1(\mathbf{p}_1, \hat{\mathbf{a}}_1)$	$\text{L}_2(\mathbf{p}_2, \hat{\mathbf{a}}_2)$	Parallel	$S_R : [\hat{\mathbf{a}}_{1\perp 1}; \hat{\mathbf{a}}_{1\perp 2}]$ $S_t : [\hat{\mathbf{a}}_{1\perp 1}; \hat{\mathbf{a}}_{1\perp 2}]$	$N_R : [\hat{\mathbf{a}}_1]$ $N_t : [\hat{\mathbf{a}}_1]$

¹ Pl = Plane, L = Line, Pt = Point, Coinc = Coincident, Dist = Distance, \mathbf{p} = Point, \mathbf{n} = Normal direction, \mathbf{a} = Axis

these DoFs. Since the tracker uses a motion model and maintains a history of the previous pose, it might be able to predict the correct pose of the object in from this partial view.

7.3.2 Geometric model of the environment constraints

We exploit knowledge about the environment and its effect on the object’s motion, by modeling the environment using the same primitive shapes models as described in Section 7.3.1. Geometric constraints can then be defined between primitive shapes in the environment (\mathcal{P}_e) and the object (\mathcal{P}_m). As an example, for an object that only moves on a tabletop, a Coincident constraint can be defined between a plane on the object’s bottom surface and the plane of the table. This restricts the translation of the object along the normal direction of the tabletop plane, and orientations along the other two orthogonal axes.

7.3.3 Constraint solver

We choose the exact geometric solver (Section 3.4) for this application. This provides several advantages over iterative approaches. Most importantly, the transformation manifolds and projection functions need to be calculated once for each frame of the scene point cloud and can be re-used for all the particles. Also, the exact solver is much faster than iterative approaches. For our constraints, the exact solver can provide runtimes of approximately $50\mu\text{s}$.

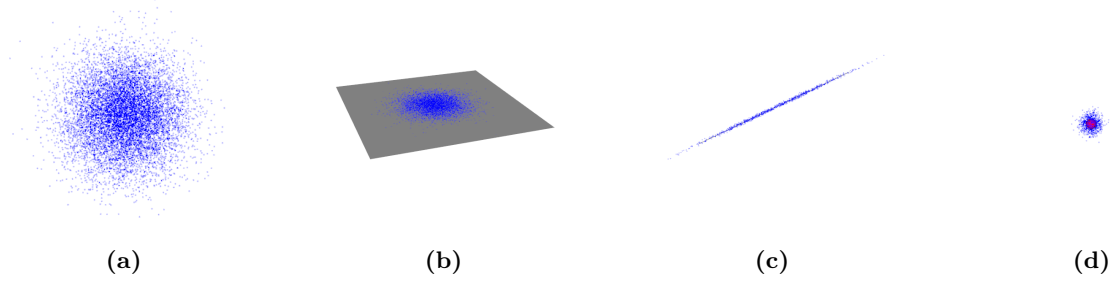


Figure 7.3: Projection of particles to the nullspace of geometric constraints (a) 3D particle set (b) Particles projected onto a plane (c) Particles projected onto a line (d) Particles projected onto a point

7.4 Adaptive Particle Filter

The Particle Filter based tracker holds a state-space representation of the 3D object. The pose is modeled in 6D (position and orientation) and represented as

$$\mathbf{s}_t = (x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t) \quad (7.1)$$

7.4.1 Projection sampling

The Particle Filter incorporates a Brownian motion model for the state prediction. As compared to a normal particle filter, in our method the Filter’s dynamic model is further influenced by the projections $f_{\text{proj}}(\mathbf{s})$ obtained from the geometric constraints detection system, as discussed in Sec 7.3 and Algorithm 11. Fig. 7.3 shows an illustration of the projection operations. In case of one plane match, the translation nullspace is a plane, and all points in the particle set s_t are projected onto the plane (Fig. 7.3(b)). For two plane matches, the constraint is a Line-Line Coincident constraint between the lines of intersection of the two sets of matched model and scene planes. The translation nullspace is a line, and all points in the particle set s_t are projected onto it (Fig. 7.3(c)). For three plane matches, the constraint is a Point-Point Coincident constraint between the point of intersection of the two sets of matched model and scene planes. The translation nullspace is a point, and all points in the particle set s_t are projected onto it (Fig. 7.3(d)).

During the prediction phase several prior state hypothesis s_t^i are generated by the Particle Filter to form the particle (sample) set using the previous particle distribution $(s^i, w^i)_{t-1}$. The motion model used is Brownian in nature: (7.2) where, w is the white Gaussian noise with defined covariance in the s_t state variables and K is the number of particles.

$$\mathbf{s}_t^i = f_{\text{proj}}(\mathbf{s})(\mathbf{s}_{t-1}^i) + \mathbf{w}_t^i, \quad i = 1, 2, \dots, K \quad (7.2)$$

7.4.2 Covariance adaptation

In the re-sampling step, the particle filter uses the estimated Gaussian noise covariance value to add random noise to the sampled particles. This Brownian model is an important feature of the particle filter approach and enables it to adapt to measurement noises. This value should closely correspond to the actual random noise in the measurement, e.g, the inherent accuracy and precision of the sensor. This

7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)

Table 7.2: Projection functions for supported geometric constraints

Fixed	Constrained	Constraint (C)	Projection Functions ($f_{\text{proj}}(s)$)	
			Translation ($\text{proj}_p(\mathbf{u})$)	Rotation ($\text{proj}_r(\mathbf{R}_i)$)
$\text{Pl}_1(\mathbf{p}_1, \hat{\mathbf{n}}_1)$	$\text{Pl}_2(\mathbf{p}_2, \hat{\mathbf{n}}_2)$	Dist. (d)	$\mathbf{u} + ((\mathbf{p}_1 - \mathbf{u}) \cdot \hat{\mathbf{n}}_1)\hat{\mathbf{n}}_1 + d\hat{\mathbf{n}}_1$	$\mathbf{R}_{AA}(\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2, \angle(\hat{\mathbf{n}}_1, \mathbf{R}_i \hat{\mathbf{n}}_2))\mathbf{R}_i$
$\text{Pl}_1(\mathbf{p}_1, \hat{\mathbf{n}}_1)$	$\text{Pl}_2(\mathbf{p}_2, \hat{\mathbf{n}}_2)$	Coinc.	$\mathbf{u} + ((\mathbf{p}_1 - \mathbf{u}) \cdot \hat{\mathbf{n}}_1)\hat{\mathbf{n}}_1$	$\mathbf{R}_{AA}(\hat{\mathbf{n}}_1 \times \hat{\mathbf{n}}_2, \angle(\hat{\mathbf{n}}_1, \mathbf{R}_i \hat{\mathbf{n}}_2))\mathbf{R}_i$
$\text{L}_1(\mathbf{p}_1, \hat{\mathbf{a}}_1)$	$\text{L}_2(\mathbf{p}_2, \hat{\mathbf{a}}_2)$	Coinc.	$\mathbf{p}_1 + ((\mathbf{u} - \mathbf{q}) \cdot \hat{\mathbf{a}})\hat{\mathbf{a}}$	$\mathbf{R}_{AA}(\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_2, \angle(\hat{\mathbf{a}}_1, \mathbf{R}_i \hat{\mathbf{a}}_2))\mathbf{R}_i$
$\text{Pt}_1(\mathbf{p}_1)$	$\text{Pt}_2(\mathbf{p}_2)$	Coinc.	\mathbf{p}_1	\mathbf{R}_i
$\text{L}_1(\mathbf{p}_1, \hat{\mathbf{a}}_1)$	$\text{L}_2(\mathbf{p}_2, \hat{\mathbf{a}}_2)$	Parallel	\mathbf{u}	$\mathbf{R}_{AA}(\hat{\mathbf{a}}_1 \times \hat{\mathbf{a}}_2, \angle(\hat{\mathbf{a}}_1, \mathbf{R}_i \hat{\mathbf{a}}_2))\mathbf{R}_i$

¹ Pl = Plane, L = Line, Pt = Point, Coinc = Coincident, Dist = Distance, \mathbf{p} = Point, \mathbf{n} = Normal direction, \mathbf{a} = Axis, \mathbf{u} = Input Point, \mathbf{R}_i = Input rotation matrix, $\mathbf{R}_{AA}(\mathbf{a}, \theta)$ = Rotation matrix defined by Axis \mathbf{a} and angle θ

value also depends on the sensor data processing that might introduce delays and therefore additional uncertainty in the actual measurement.

In our adaptive particle filter, we exploit the information about primitive shapes detected in the scene to adapt this noise estimate. The primitive shape detection algorithm provides estimated noise in the detection process (e.g. average fitting error for points on a plane). The default noise covariance vector \mathbf{w}^{def} represents the estimated noise in the raw sensor data. For the combined primitive shape constraint used for tracking \mathbf{C}_{comb} , the estimated noise in position and orientation is σ_R and σ_t respectively. A constraint specifies the directions it constrains (Controlled Space Directions) and the directions where it has no effect (Nullspace directions), as listed in Table 7.1. In the directions controlled by the constraint (Controlled Space Directions) the estimated Gaussian noise covariance corresponds to σ_R and σ_t , while in the remaining directions (Nullspace directions) the estimated Gaussian noise covariance is \mathbf{w}^{def} . Equation (7.3) combines these to generate the adapted covariance vector $\hat{\mathbf{w}}$.

$$\hat{\mathbf{w}} \leftarrow [(\mathbf{N}_t \cdot \mathbf{w}_t^{\text{def}})\mathbf{N}_t + (\mathbf{S} \cdot \sigma_t)\mathbf{S}_t, (\mathbf{N}_R \cdot \mathbf{w}_R^{\text{def}})\mathbf{N}_R + (\mathbf{S}_R \cdot \sigma_R)\mathbf{S}_R] \quad (7.3)$$

7.4.3 Hypothesis verification

For every particle hypothesis the object model is projected into the point cloud scene I and a likelihood is computed w.r.t. a distance matching metric. It requires evaluation of a distance measure D_p between the projected point cloud of the object model and the corresponding nearest neighbor points in the measurement P_i . This is followed by the computation of the distance measure D_c on the color separation of the two point sets.

$$D_p = \frac{1}{N} \sum_{n=1}^N \frac{\text{dist}(\mathbf{m}_{xyz}^n, \mathbf{I}_{xyz}^n)(1 - \mathbf{m}_{\text{normal}}^n \cdot \mathbf{I}_{\text{normal}}^n)}{r} \quad (7.4)$$

Equation (7.4) presents the first metric where, N is the number of projected points based on the camera perspective (only points visible to the camera view are considered by analyzing the normals),

$dist(\mathbf{m}, \mathbf{I})$ refers to the distance between a model point \mathbf{m} and the approximated nearest neighbor point in the scene within a search radius r . The computed distance is further biased by the angle between the normal directions of the model point and the matched scene point. This is obtained using the dot product between the two normals wherein a better match reduces the overall distance measure. (7.4) represents the metric.

$$\begin{aligned} \mathbf{q}_m &= f_{\text{histo2D}}(m_{\text{hsv}}) \\ \mathbf{q}_I &= f_{\text{histo2D}}(I_{\text{hsv}}) \\ D_c &= \left[1 - \sum_{n=1}^N \sqrt{\mathbf{q}_m(n) \mathbf{q}_I(n)} \right]^{1/2} \end{aligned} \quad (7.5)$$

Equation (7.5) shows the color distance measure which is obtained by generating the joint probability histograms [124] of the projected model points and the matched scene points obtained from the nearest neighbor search as a part of the point matching distance measure. The RGB values of the points are first converted to a HSV color space and thereafter the the joint probability histograms \mathbf{q}_m and \mathbf{q}_I are computed for the projected model and scene points respectively. In the next step a distance between the two histograms is computed using the Bhattacharya distance metric [125]. The two distance measures are fused in a likelihood function under a Gaussian model wherein a weighted sum of the the two distance measures is applied. A parameters μ_p and μ_c define the fusion weight of the point and color matching metrics respectively such that $\mu_p + \mu_c = 1$. Equation (7.6) presents the likelihood model where D_f is the fused distance measure and λ is the Gaussian likelihood model covariance.

$$\begin{aligned} P(z|s_t^i) &= \exp\left(-\sum_{p=1}^K (D_f^2/\lambda)\right) \\ D_f &= \mu_p D_p + \mu_c D_c \end{aligned} \quad (7.6)$$

Based on the computed likelihood, the weight w_i of each particles is updated. Based on the normalized weights distribution a importance based re-sampling strategy [126] is applied to the particle set.

7.4.4 Complexity

Applying the Particle Filter for tracking objects in the scheme mentioned above is challenging in terms of computational complexity arising from a very high number of particles required to track the object in 6D. The point cloud based likelihood models themselves are computationally heavy give their big data nature. Furthermore, sensor noise, scene clutter and free dynamics of the object effects the stability of the tracker resulting in tracking loss. In our work we precisely address this problem of Particle Filters through geometric constraints. We define a finite parameter set which influences the sub search space of the Particle Filter. It is gives as follows:

$$PF_{\text{params}} = \left\{ \begin{array}{ll} K, & \text{particle count } K > 0 \\ w, & \text{process noise covariance} \\ \mu_c, & \text{colour modality weight} \\ & 0 \leq \mu_c \leq 1 \end{array} \right\} \quad (7.7)$$

On startup, the Particle Filter is initialized using a rough pose of the object using a standalone detector. This information is sufficient to obtain a initial estimate of the object's state. However, to

7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)

maintain robust track of the object which exhibits free dynamics, the parameter set represented in (7.7) requires one-line adaptation in real-time. In order to achieve this the companion geometric constraint detection system operates in close synchronization with the Particle Filter. At each time step, the constraint detection system gets the current estimated pose of the target from the Particle Filter. Based on the association between the target pose and the geometric constraints detected (plane, line or point) a new set of parameters are generated for the Particle Filter. This process essentially locks one or more degrees of freedom of the update model of the Particle Filter which is reflected in the new covariance set for the dynamic update model. In addition the reduction in the state space dimension allows reduction in the particle count resulting in reduced computational complexity of the filter. Furthermore, it is observed that the color modality allows convergence of the Particle Filter when the filter initialization is considerably away from the ground truth or under tracking loss scenarios. In such circumstances the influence of the color modality can be enhanced using the μ_c parameter. Fig. 7.6 compares the performance of these different tracker modes in cases of tracking loss.

7.5 Experiments and Evaluations

We conducted several experiments, both in simulated scenes and on real data from a Kinect v2 sensor. Through these experiments, which are described in detail in the following subsections, we show how objects can be accurately tracked by our system with very few particles.

7.5.1 Experimental setup

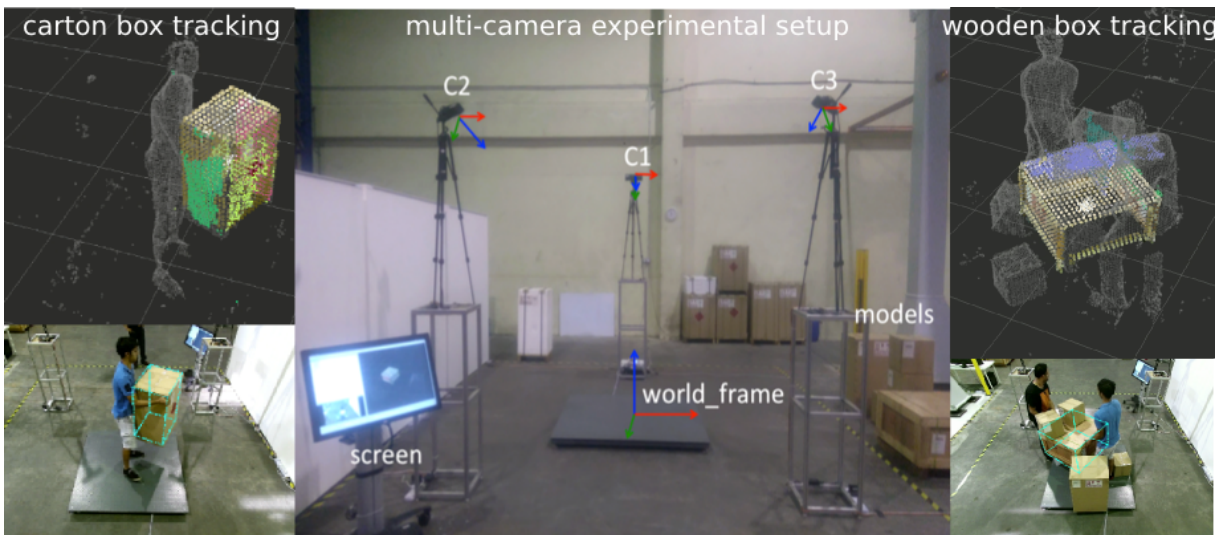


Figure 7.4: Experimental setup with illustrations of some live tracking scenarios

Our live experimental setup (Fig. 7.4) consists of 3 NUC/Kinect units ($C1, C2, C3$) with a tracking space of approximately (3 m x 3 m x 2.5 m). RGB and Registered Depth images are extracted via the libfreenect2 and iai-kinect2 [127] drivers and sent via the ROS (Robot Operating System) interface to a central computer. The 3 cameras are extrinsically calibrated to a central WorldFrame. We then create a fused streaming point cloud by generating, transforming and fusing the points together into the same

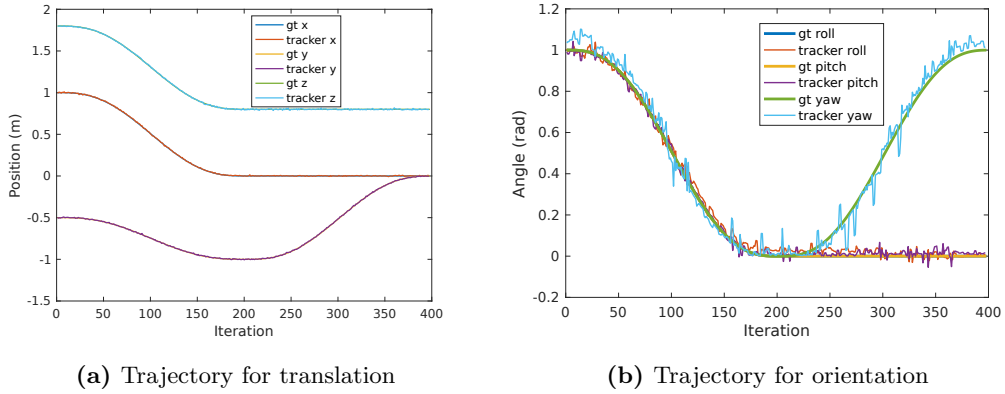


Figure 7.5: Tracking trajectories for object JuiceBox using TPSC

frame on the GPU. A voxel grid filter followed by polynomial fitting [128] is used to filter out noise, and finally generate data $(r, g, b, x, y, z, n_x, n_y, n_z)$ at approx. 10Hz. We run our experiments on a Intel 6700K CPU and a Geforce GTX 1070.

7.5.2 Evaluation of TPSC

For evaluation, we used the set of models available from [109]. The objects and their simplified models based on primitive shape are shown in Fig 7.2.

We quantitatively examined several aspects of our tracking approach. Firstly, we evaluate the effect of color information (controlled by parameter μ_c) on the tracking performance. Secondly, we determine the benefits of using primitive shape constraints. For each object in the evaluation, we used 4 different tracking approaches: Only 3D (i.e., using only 3D point cloud information D_p), color+3D (i.e., $\mu_p D_p + \mu_c D_c$), 3D+PSC (D_p with primitive shape constraints), and the full TPSC (i.e. $\mu_p D_p + \mu_c D_c$ with primitive shape constraints).

For the tracking scenario, we used a tabletop scene with the object. In the first segment of the ground truth trajectory, the object is moved in all 6 DoFs until it reaches the table. In the second segment, the object moves along the table with 3 DoFs (translation along x and y axes, and rotation along z-axis). The results showing the errors in each axis for all the object and tracker combinations are summarized in Table 7.3. Tracking trajectories in rotation and translation for the object JuiceBox using the full TPSC tracker over the simulated dataset are shown in Fig. 7.5. Fig. 7.6 compares the evolution of tracking errors from an initial position for the 4 different trackers. It is clear that the full TPSC tracker shows the best convergence properties.

From the evaluation, and a comparison with the results in [109], it is clear that our approach can achieve reasonably accurate tracking by use of very few particles. In most cases, use of color information improves the tracking, but there are a few outliers. This is more prominent for the Box, and the probable reason is a low-resolution color model and similar textures on all faces of the box. In particular, it can be observed that tracking with use of primitive shapes always improves the tracking accuracy, and requires lesser particles.

7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)

Table 7.3: Evaluation of TPSC

Object	Tracker	#Particles (min. – avg. – max.)	RMS Errors (mm)			RMS Errors (deg)		
			X	Y	Z	Roll	Pitch	Yaw
Box	Only 3D	500	10.2	8.7	7.9	7.7087	6.1825	11.6113
	color+3D	500	10.0	9.8	7.6	6.0418	6.0899	10.3905
	3D+PSC	150 – 255.7 – 500	10.6	3.8	3.3	1.7763	1.8372	2.5001
	color+3D+PSC	150 – 255.1 – 500	8.7	3.3	5.0	1.7898	1.7909	3.1615
Juice	Only 3D	500	15.4	11.8	6.8	7.9867	7.1782	9.8036
	color+3D	500	13.4	8.3	6.2	8.0417	8.0357	8.4454
	3D+PSC	150 – 335.2 – 500	12.0	3.7	6.1	1.9949	3.0015	4.0498
	color+3D+PSC	150 – 304.3 – 500	3.9	3.9	4.8	1.4417	1.0215	2.6449
Milk	Only 3D	500	10.7	6.0	8.2	3.9158	4.3284	7.2542
	color+3D	500	8.4	7.2	6.7	6.1082	4.8459	8.4107
	3D+PSC	150 – 379.8 – 500	5.9	4.6	7.1	3.2659	3.1813	4.2037
	color+3D+PSC	150 – 340.3 – 500	5.0	4.0	5.8	3.1820	2.6500	3.7363
KinectBox	Only 3D	500	10.4	23.2	14.1	4.9606	6.1498	5.0923
	color+3D	500	10.4	16.9	12.7	4.7392	7.0245	5.3812
	3D+PSC	150 – 342.5 – 500	9.3	19.2	9.7	3.6295	3.3586	3.3702
	color+3D+PSC	150 – 355.2 – 500	5.9	10.4	7.4	1.9650	3.4623	1.7482

7.5.3 Application in logistics domain

We use some examples from the logistics domain to evaluate our tracking approach. This domain is very relevant for our approach since a large fraction of cargo items are comprised of simple shapes such as cuboids and cylinders. We show a tracking scenario with 3 kinect V2 sensors, where some cargo items are tracked (Fig. 7.7).

We use simulated object trajectories to generate a synthetic dataset and use this to compare two trackers under different settings. Our baseline tracker (“Only 3D”) is a simple particle filter that doesn’t use the primitive shape information. The second tracker is our primitive shape based tracking approach (“3D+PSC”) described in Section 7.4. We quantitatively evaluate several properties of the trackers. We study the effect of varying the number of particles on the accuracy of tracking, i.e. the RMS tracking errors in rotation and translation w.r.t the ground truth. The results of this experiment are summarized in Table 7.4. From this evaluation, we can see that the use of primitive shape constraints improves the tracking error. In some cases, the performance of the “3D+PSC” tracker with 50 particles is close to or better than the “Only 3D” tracker with 200 particles. This confirms our expected behavior that the use of primitive shapes reduces the search subspace and hence, the required number of particles to sample the space.

Table 7.4: Evaluation of TPSC in a logistics scenario

Object	Tracker	#Particles	RMS Errors (mm)			RMS Errors (deg)		
			X	Y	Z	Roll	Pitch	Yaw
WoodBox	Only 3D	50	26.1	23.9	27.4	2.91	2.79	2.75
	Only 3D	100	21.4	25.0	23.2	2.11	2.29	1.82
	Only 3D	200	14.9	19.5	19.7	1.45	1.53	1.35
	3D+PSC	50	14	14.1	16.3	1.69	1.57	1.40
	3D+PSC	100	14	15.2	13.6	1.21	1.29	1.08
	3D+PSC	200	10.1	12.7	14.3	1.097	0.87	0.85
Box	Only 3D	50	15.4	17.3	56.2	5.74	19.05	7.36
	Only 3D	100	11.8	24.6	23.9	1.97	1.65	2.00
	3D+PSC	50	14.3	12.6	34.0	5.01	18.87	7.13
	3D+PSC	100	10.3	19.4	16.1	1.25	1.07	1.28

We also plot the tracking errors over time for this dataset for the object WoodBox. Fig. 7.9 shows the results for this experiment, where tracking errors for the “3D+PSC” tracker in blue are lower than that of the “Only 3D” tracker throughout the tracking sequence.

In addition to the synthetic data, we also performed several experiments with real sensor data captured using our experimental setup. Some images of these experiments showing the properties of our tracker are shown in Fig. 7.7. We compared the tracking trajectories for these two solvers on this database. Fig. 7.8 shows this comparison, where the trajectories for the “3D+PSC” tracker seem more smooth compared to those for the “Only 3D” tracker.

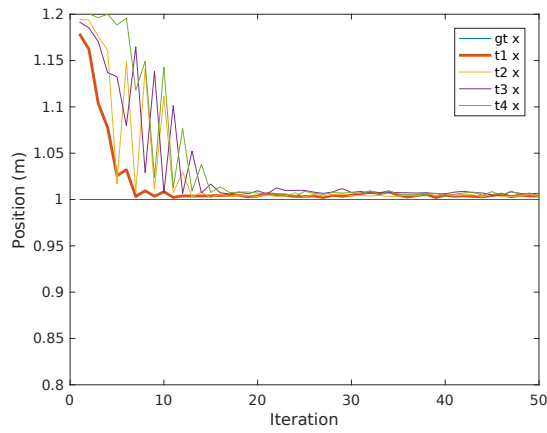
More results from this application are presented in [10] and Appendix B.

7.6 Discussion

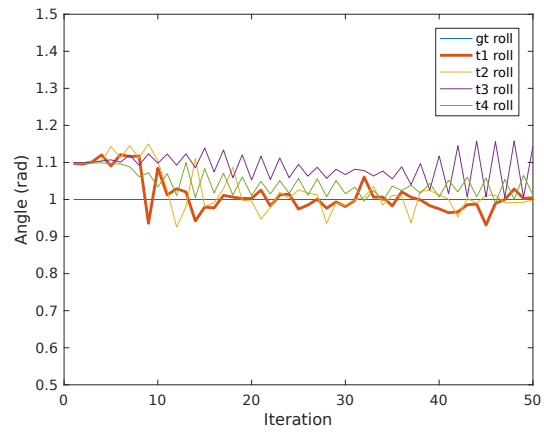
In this Chapter, we have presented an approach for exploiting geometric information about the object and its environment to improve the performance of a Particle Filter based tracker. This is achieved by combining an intelligent primitive shape detection component that can also estimate the shape fitting error, and an exact geometric constraint modeling framework that can generate the geometric nullspace with bounds based on these errors. An adaptive Particle Filter formulation restricts its search to this nullspace, and also adapts the process covariances according to the estimated fitting error.

In principle, our approach can be used in higher dimensional spaces and is not specialized for 6D space. We focused our experiments towards 6D rigid objects, but this can be extended to articulated objects (e.g. hands, skeletons) in the future. Also, in many practical tracking scenarios the tracker complexity is also influenced heavily by the number of objects to be simultaneously tracked. If each object requires a smaller (and lower dimensional) search subspace based on our method, more objects can be tracked simultaneously with the same number of particles.

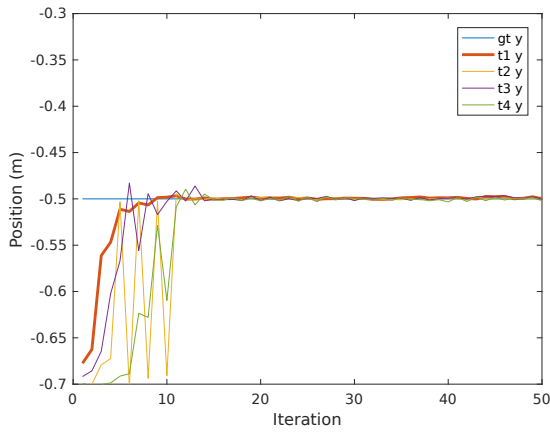
7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)



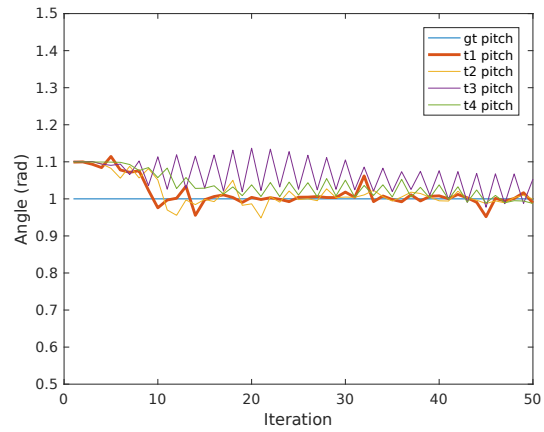
(a) Translation X



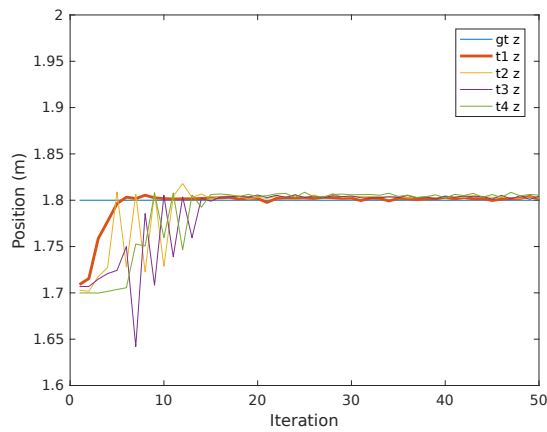
(b) Rotation Roll



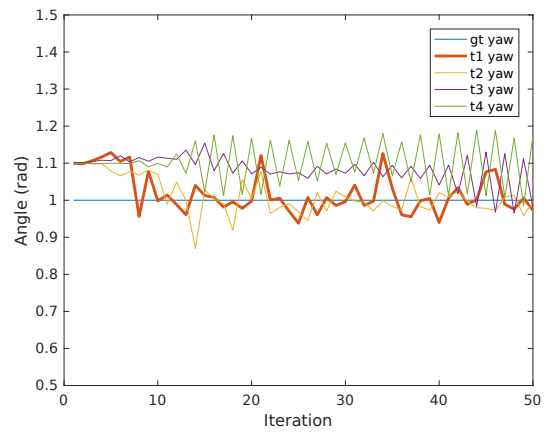
(c) Translation Y



(d) Rotation Pitch



(e) Translation Z



(f) Rotation Yaw

Figure 7.6: Convergence of trackers from an initial pose, for the object Juice. The four trackers mentioned in Table 7.3 have been tested: t1 refers to “color+3D+PSC”, t2 refers to “3D+PSC”, t3 refers to “color+3D”, t4 refers to “Only 3D”.

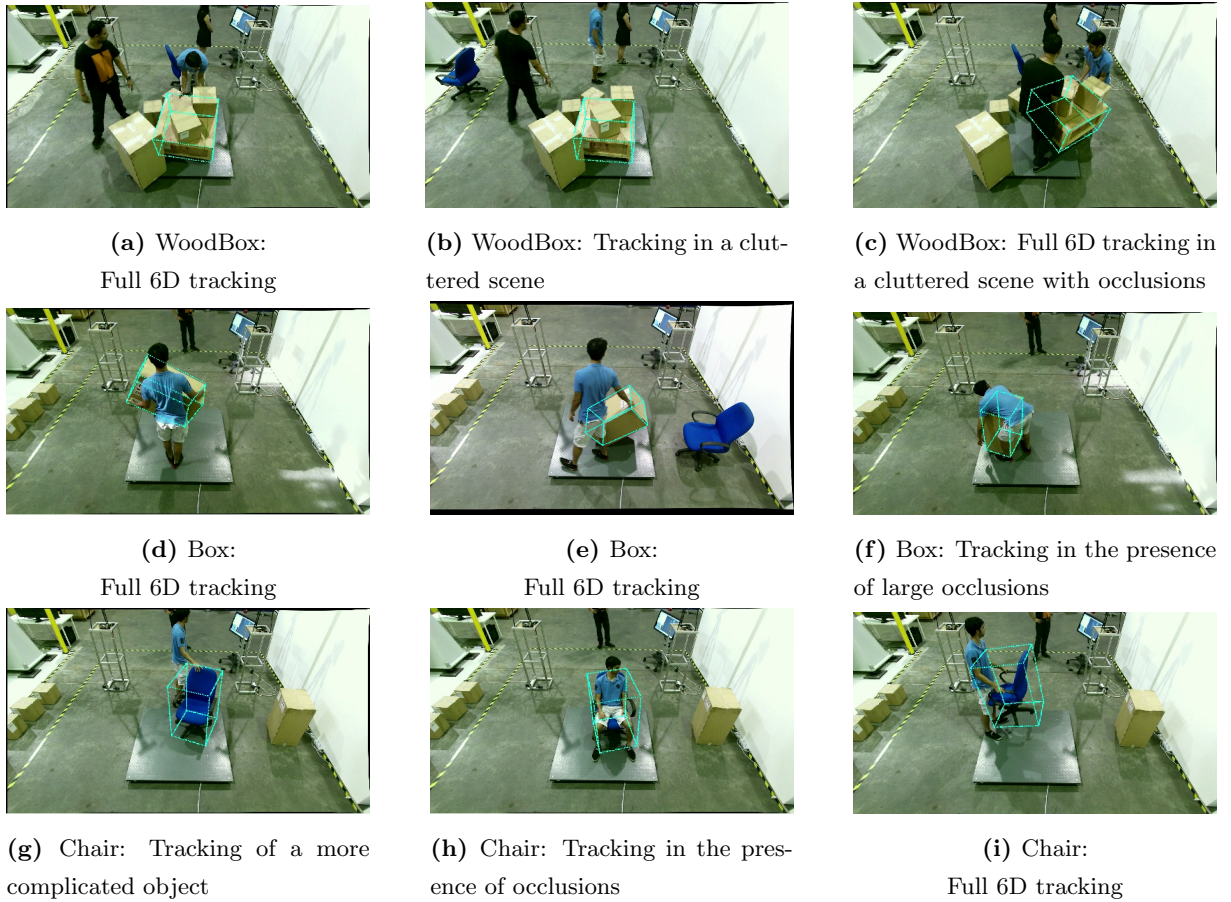


Figure 7.7: Constraint-based tracking for 3 objects in our experimental setup.

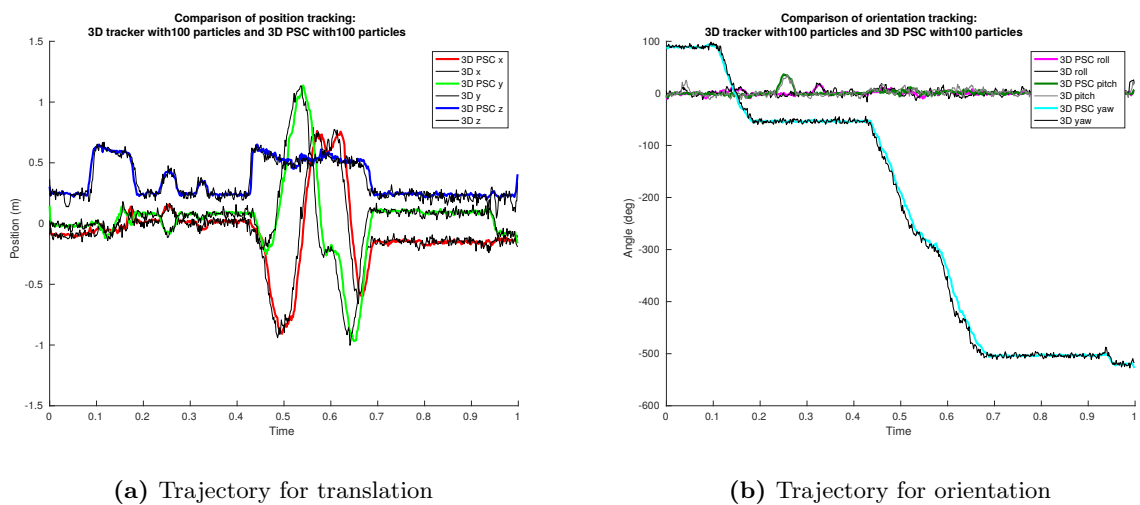
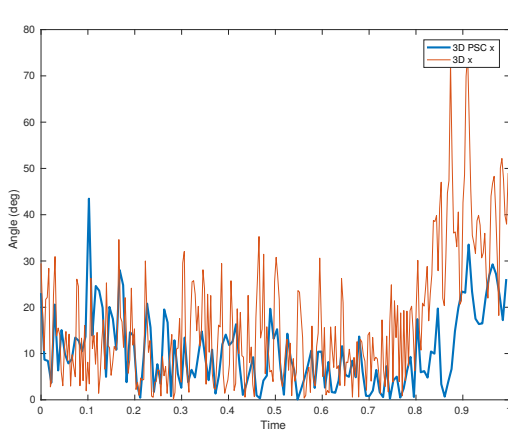
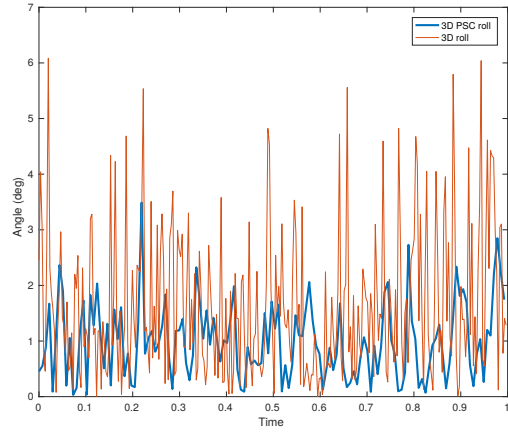


Figure 7.8: Comparison of trajectories for object WoodBox using our live experimental setup

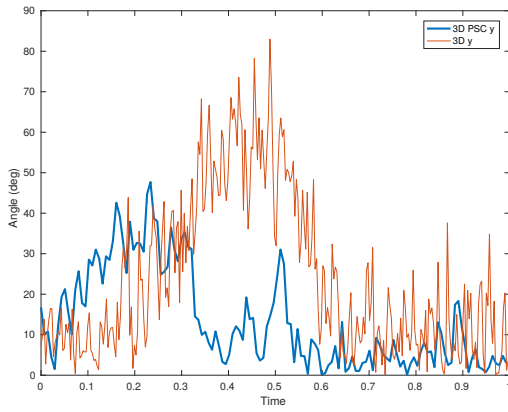
7. TRACKING USING PRIMITIVE SHAPE CONSTRAINTS (TPSC)



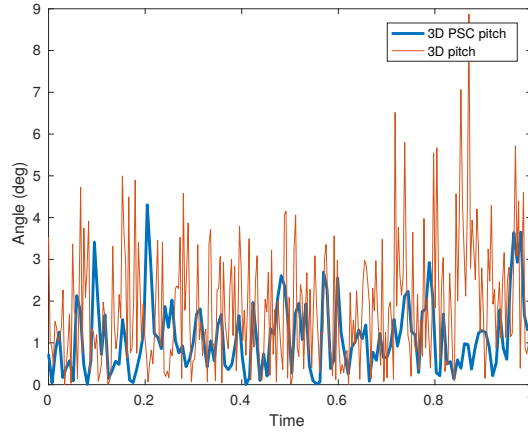
(a) Translation X



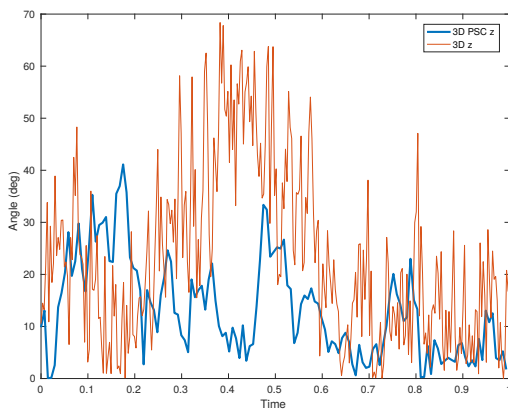
(b) Rotation Roll



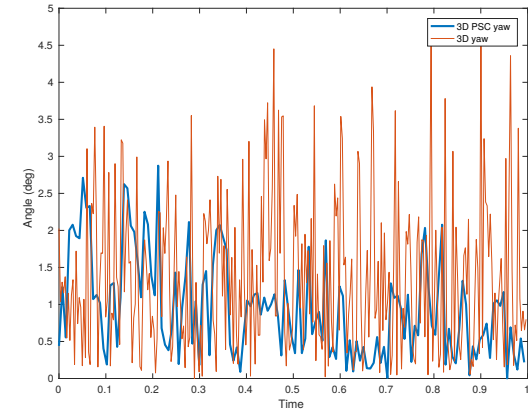
(c) Translation Y



(d) Rotation Pitch



(e) Translation Z



(f) Rotation Yaw

Figure 7.9: Comparison of tracking errors on synthetic data from a pre-defined trajectory for the object WoodBox.

Part IV

Conclusion and Appendix

Chapter 8

Conclusions and Future Work

In this work, we presented the first steps towards a complete constraint-based approach for robotic systems. While we have tried to provide sufficient evidence that this is not only feasible, but also provides improvements over state-of-art methods, there are several concepts that need to be studied in more depth and many extensions to this work are possible.

8.1 Contributions

The main contribution of this work is the use of constraints to describe diverse classical problems in robotic systems. Constraint-based methods have been used in several fields, especially in the development of robot motion controllers. However, to the best of our knowledge, constraint-based methods haven't been applied to classical computer vision problems of model-based object detection, pose estimation and tracking. Our unifying constraint modeling and constraint solving framework for handling such diverse problems is the core contribution of this work. Through four classical problems, we have tried to show that a constraint-based approach for modeling intelligent robotic systems is feasible. Fig.8.1 shows a collection of some key images that illustrate our contributions to the different problems.

8.1.1 Constraint modeling and solving framework

Our solution is based on our own constraint modeling and solving framework. One key observation in focusing the direction of this work is the fact that geometry and geometric constraints are common elements in each of these fields. Hence, we focus on modeling and solving geometric constraints. We have proposed an exact geometric constraint solver that presents several advantages over the state-of-art. Firstly, it supports constraints with inequalities. Secondly, it supports mixed transformation manifolds, i.e. cases where the rotation and translation elements in the constraint are dependent. We have provided several examples to prove the importance of these extensions. We developed an iterative solving approach that can handle non-linear constraints with inequalities and is necessary for modeling environment and optimization constraints. We also presented a hybrid solver that combines the advantages of exact and iterative solvers: incorporating the powerful and generic constraint model from iterative solvers, while exhibiting better convergence properties with more predictable runtimes due to better initialization using the exact solver.

8. CONCLUSIONS AND FUTURE WORK

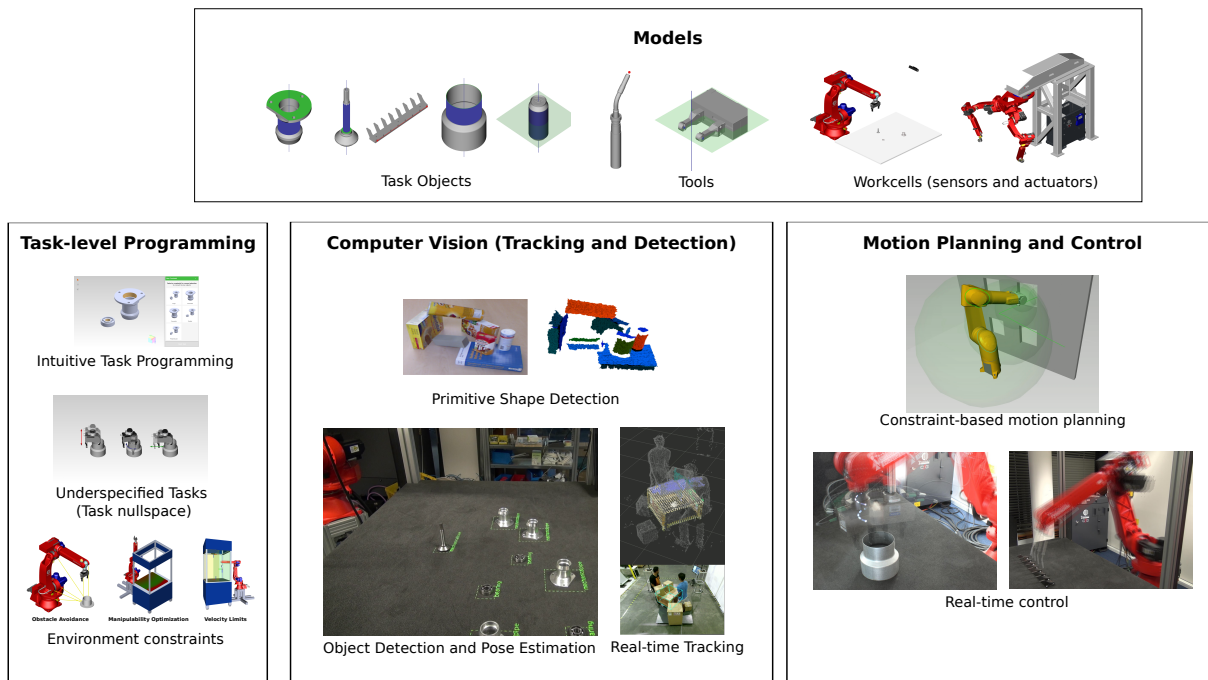


Figure 8.1: Overview of contributions: Our modeling approach is key to all the applications. The three main application domains are: Task-level robot programming, Computer Vision (Model-based detection and tracking), and Robot Motion Planning and Control.

8.1.2 Task level robot programming, planning and real-time control

We have developed a constraint-based robot task description methodology based on our constraint modeling and solving frameworks. This framework can be used for several levels of robot task description: CAD-like mating interface for creating assemblies, offline programming of robot motions, and real-time control with environment constraints. While subsets of the features that our framework supports are also possible with other state-of-art frameworks, our complete set of features is unique. Also, the scope of geometric constraints supported by our framework goes beyond the state-of-art. We have also developed a constrained motion planning approach that combines a probabilistic motion planner with our exact geometric solver. The constraint-based task and goal descriptions can be directly used in this motion planner to generate collision-free paths while satisfying task constraints.

8.1.3 Computer vision

We view the classical computer vision problem of object detection and pose estimation from a different and uncommon perspective. We formulate the pose estimation problem as a constraint solving problem. Our method is specialized for objects that lack any distinctive texture, and can be decomposed into a small set of primitive geometric shapes such as planes, cylinders and spheres. Our method detects such primitive shapes in a given scene, matches them to primitive shapes in the object model. Each such match adds a geometric constraint on the pose of the model's instance in the scene. We present efficient RANSAC-based algorithms to generate object instance hypotheses. Our method also presents a systematic approach to handle symmetric objects with less than 6 DoFs.

In the field of model-based visual tracking, adaptive particle filters are among the most powerful and commonly used algorithms. The dimensionality of the tracking space, and consequently its size is a major factor that affects the tracker performance. This is simply due to the reason that the number of particles required to effectively cover the tracking space increases exponentially with the number of dimensions. We propose to reduce the dimensionality and size of this tracking subspace by exploiting information of primitive shapes in the object models. Similar to our object detection module, we detect primitive shapes in the scene and associate them with object instance hypotheses. Each such match adds a geometric constraint to the pose of the object instance in the scene. The tracker then only need to track the objects in the nullspace of these geometric constraints. Through several experiments, we show that compared to a standard particle filter, our constraint-based tracking approach achieves comparable accuracy with significantly fewer particles.

8.1.4 Demonstrations

Our framework was used in the development of intuitive interfaces for programming industrial robots, which were used in several industrial robotic applications such as assembly, woodworking and welding. We applied our method to several robotic platforms including several 6 DoF manipulators, redundant manipulators such as the 7 DoF Kuka light weight arm, a 13 DoF dual arm robot, and a large gantry robot. We conducted extensive simulations to evaluate our framework, and also used it on real robots to prove its capabilities in practical applications. Our results were used in several demonstrators in a European robotics research project SMErobotics¹, and were also presented at international trade fairs.

8.2 Shortcomings

Some of the shortcomings and known potential issues with our approach are presented in the following sections.

8.2.1 Further extension of exact solver

In this work, we extended the exact solver from Rodriguez [11] to include constraints with inequalities and mixed transformation manifolds. It significantly increased the complexity of the solver in terms of the number of constraint combination rules and manifold mappings. Another extension for this solver is to support more powerful geometric descriptors such as Non-uniform rational basis splines (NURBS)². In our opinion, this extension is not trivial. The key idea for the exact solver is based on the fact that closed form solutions exist to many of the constraints involving primitive geometric shapes such as points, lines, planes, cylinders, spheres, etc. In case of geometric descriptors such as NURBS, this is not true in the general sense. Hence, a solver to support such geometric descriptors might not be exact.

8.2.2 Runtime performance of object detection

We have approached the object recognition and pose estimation problem from the direction of primitive shapes and their properties. The algorithm is technically sound and shows good potential for the datasets

¹<http://www.smerobotics.org>

²https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

8. CONCLUSIONS AND FUTURE WORK

that we work with. However, this is a classical field in computer vision and there are many approaches that view this problem from a completely different perspective. One such method is template matching, where a large number of templates of the object are generated from different viewpoints of the object from real or synthetic data. This template is then matched to the real scene during runtime to detect objects and estimate their pose. This method requires a significantly larger amount of memory compared to our method but the runtime performance is much better. Another such method is based on keypoints. While our approach tries to focus on the geometric elements that describe a large fraction of the object model, these methods tend to focus on specific keypoints that form a small fraction of the object model but are most discriminative. Intuitively, our approach tends to work better in situations where the object model is very simple and prone to occlusions and partial views. The keypoint-based approach works better in cases where the object has distinctive points that are also well spread across the object. The key advantage of keypoint-based methods over ours is their speed.

In conclusion, we believe that our method is suitable for a set of objects and scenarios where the traditional approaches might not work at all. The runtime performance is an issue with our method, but we believe that this can be solved in the future with efficient implementations and hardware acceleration. We have used parallelization on the CPU with OpenMP, but haven't explored GPU implementations yet. This could result in a major speed-up and alleviate these concerns.

8.2.3 Dependence on detection of primitive shapes

We have presented our own approach for detection of primitive shapes from point cloud data. However, our focus has been on the use of these detected primitives for object detection or tracking. Current implementations of primitive shape detectors, both ours and state-of-art, are rather slow and significantly affect the overall runtime of our algorithms. This is particularly the case for primitives such as cylinders or spheres. Efficient approaches for detecting these primitives from sensor data are essential for our constraint-based detection and tracking algorithms to be effective.

8.3 Proposed Future Work

There are several directions for future work on the topics presented in this thesis. There are several possible extensions to the constraint-based modeling framework. Also, we believe that our work can be useful in many other classical problems in the fields of computer vision and robotics. Some of these ideas are presented in the following Sections.

8.3.1 Convergence

In Section 1.6, we presented the concept of a robotic system architecture that combines the different problem domains that we covered in this work. However, within the limits of this thesis, we have treated each domain separately. Some more effort needs to be directed towards conducting experiments that can combine these applications together and exploit the synergy effects. As an example, the estimated errors and tolerances from vision can be used to define compliance in control.

8.3.2 Tracking in higher dimensions

The constraint-based tracking approach that we presented is well suited for high-dimensional tracking. In this work, we only presented the basic ideas with proof-of-concept experiments. We have shown how our tracker scales well with the dimensions. In the future, we plan to do more experiments involving simultaneous tracking of multiple objects to prove this. Also, we want to evaluate our tracker on high DoF articulated objects such as human hands [129] or skeleton [130].

8.3.3 Tracking and detection threads

A common concept in computer vision systems is the use of detection and tracking in parallel threads. The detector in this case is a completely state-less thread, where each sensor frame is treated without any bias from previous states. The tracker requires initialization, and is prone to problems such tracking loss and getting stuck in local minima. Even with adaptive particle filters with random restarts, a large number of particles or iterations might be necessary to recover from such situations. This detection thread is used for initializing or correcting the tracker in such situations. We presented our detector and tracker as separate modules and evaluated them in separate experiments. In the future, we plan to combine these.

8.3.4 Combined motion and force control

Our real-time controller focused on position and velocity level constraints. This was also motivated by the fact that most industrial robot control interfaces support position or velocity control only. Extending this approach to force control is also a direction of future work. We made some progress in this direction in the work by Cai et al. [41] [131]. We could extend our hybrid constraint solver, where geometric constraints are solved exactly and combined with environment constraints as well force constraints in the iterative solver.

8.3.5 Constraint-based visual servoing

Cai et al. [131] presented an idea for combining Image-based Visual Servoing [132, 133, 134] with constraint-based control. While Cai et al. presented this with implementations of whole body control approaches such as WBCF [28] and TSID [40], we can extend this to use our iterative or exact solvers. Agravante et al. [135] also presented a similar idea. Studying how geometric constraints can be better used in Visual Servoing, and using our hybrid solver in this context is a topic for future work.

8.3.6 Extending our constraint controller for highly redundant robots

We have focused our experiments largely on industrial robots and manipulators with 6 DoFs. Our approaches are particularly useful in such robots since the scope for optimization in the robot joint space is rather limited and our efficient constraint solving techniques are necessary. We also performed some experiments with redundant robots such as the 7 DoF Kuka LWR and a Comau dual arm robot. The results for these robots are also promising and indicate that our method is suitable for such robots. In fact, the convergence properties for our solvers were much better in case of these robots.

8. CONCLUSIONS AND FUTURE WORK

In the future, we plan to perform more experiments with redundant robots and find applications in highly redundant robots such as humanoids. Typical approaches for such robots focus on local linearization using Jacobians, or iterative solvers. Our exact and hybrid solvers could prove beneficial for such applications by providing better initial positions and improving the convergence properties of existing approaches based on iterative solvers.

Appendix A

Transformation Utilities

A.1 Rotation defined by an axis-angle pair

Given a vector $\hat{\mathbf{u}} = (u_x, u_y, u_z)$ and an angle ϕ , the rotation matrix defining the rotation of an angle ϕ around the vector $\hat{\mathbf{u}}$ is given by A.1.

$$R_{AA}(\hat{\mathbf{u}}, \phi) = \begin{pmatrix} au_x^2 + c_\phi & au_x u_y - u_z s_\phi & au_x u_z + u_y s_\phi \\ au_x u_y + u_z s_\phi & au_y^2 + c_\phi & au_y u_z - u_x s_\phi \\ au_x u_z - u_y s_\phi & au_x u_y + u_z s_\phi & au_z^2 + c_\phi \end{pmatrix} \quad (\text{A.1})$$

where $a = 1 - c_\phi$, c_ϕ is the cosine of the angle ϕ , and s_ϕ is the sine of the angle ϕ

A.2 Rotation defined by two pairs of vectors

Given two pairs of corresponding vectors, $((\hat{\mathbf{a}}, \hat{\mathbf{b}}) \rightarrow (\hat{\mathbf{d}}, \hat{\mathbf{e}}))$ the complete rotation matrix can be calculated. This is based on the definition of rotation matrices based on orthonormal basis sets.

The orthogonal basis $(\hat{\mathbf{a}}, \hat{\mathbf{b}}_a, \hat{\mathbf{c}})$ can be calculated given two of the vectors $(\hat{\mathbf{a}}, \hat{\mathbf{b}})$ using A.2 and A.3

$$\hat{\mathbf{b}}_a = \frac{\hat{\mathbf{b}} - (\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})\hat{\mathbf{a}}}{|\hat{\mathbf{b}} - (\hat{\mathbf{a}} \cdot \hat{\mathbf{b}})\hat{\mathbf{a}}|} \quad (\text{A.2})$$

$$\hat{\mathbf{c}} = \hat{\mathbf{a}} \times \hat{\mathbf{b}}_a \quad (\text{A.3})$$

After computing the requisite orthonormal basis sets $(\hat{\mathbf{a}}, \hat{\mathbf{b}}_a, \hat{\mathbf{c}})$ and $(\hat{\mathbf{d}}, \hat{\mathbf{e}}_d, \hat{\mathbf{f}})$, the rotation matrix

$$\mathbf{R}_{vv}((\hat{\mathbf{a}}, \hat{\mathbf{b}}) \rightarrow (\hat{\mathbf{d}}, \hat{\mathbf{e}})) = [\hat{\mathbf{d}}, \hat{\mathbf{e}}_d, \hat{\mathbf{f}}][\hat{\mathbf{a}}, \hat{\mathbf{b}}_a, \hat{\mathbf{c}}]^T \quad (\text{A.4})$$

A. TRANSFORMATION UTILITIES

Appendix B

Evaluation of Tracking with Primitive Shape Constraints

In this Chapter, we present some additional evaluations for our constraint-based visual tracking approach. We use two objects, i.e., the WoodenBox and Box, for this evaluation. We use data recorded from our experimental setup as well as synthetic data with known ground truth for these experiments. The experiments with real data show the performance of the tracker in real applications with noise and clutter. The simulated data is useful for benchmarking the accuracy of the tracker w.r.t. perfectly known ground truth.

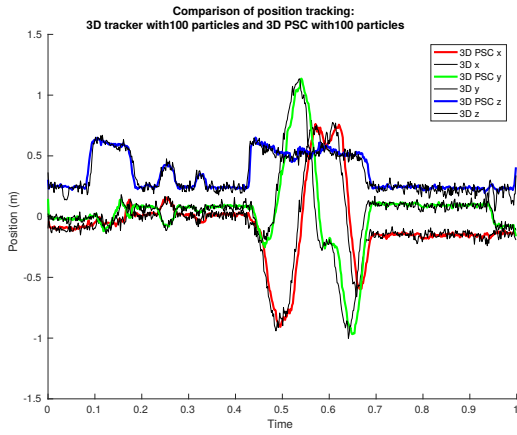
B.1 Tracking results for object WoodBox

Fig. B.1 shows the trajectories estimated by our tracker on a dataset recorded from our live experimental setup for the object WoodBox. Fig. B.2 shows the comparison of trajectories estimated by our tracker and the known ground truth on a synthetic dataset for the object WoodBox.

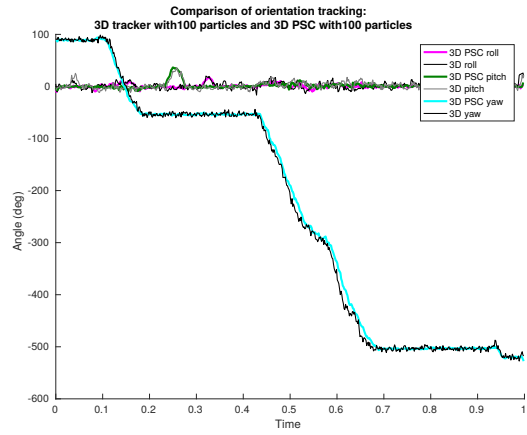
B.2 Tracking results for object Box

Fig. B.3 shows the trajectories estimated by our tracker on a dataset recorded from our live experimental setup for the object Box. Fig. B.4 shows the comparison of trajectories estimated by our tracker and the known ground truth on a synthetic dataset for the object Box.

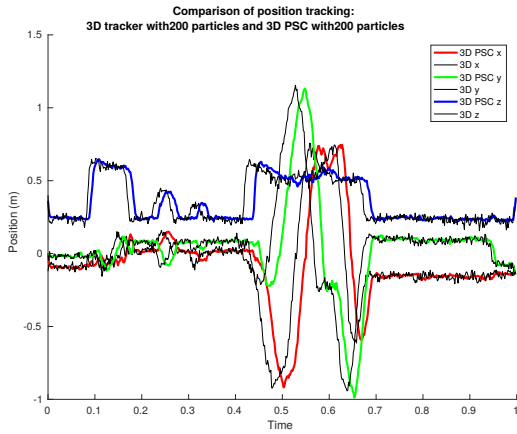
B. EVALUATION OF TRACKING WITH PRIMITIVE SHAPE CONSTRAINTS



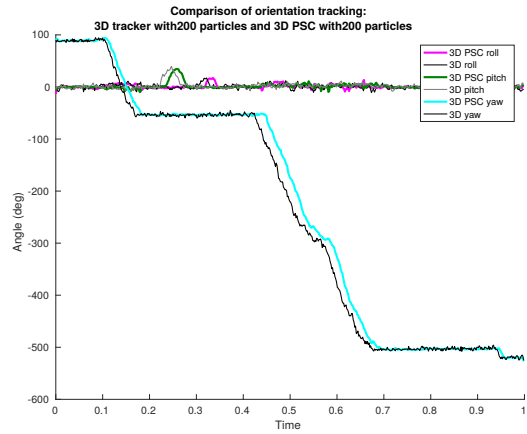
(a) Trajectory for translation (100 particles)



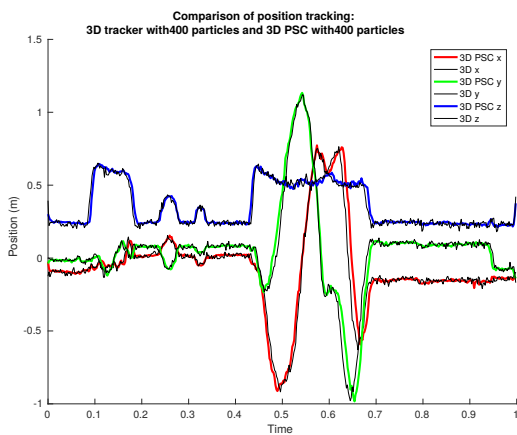
(b) Trajectory for orientation (100 particles)



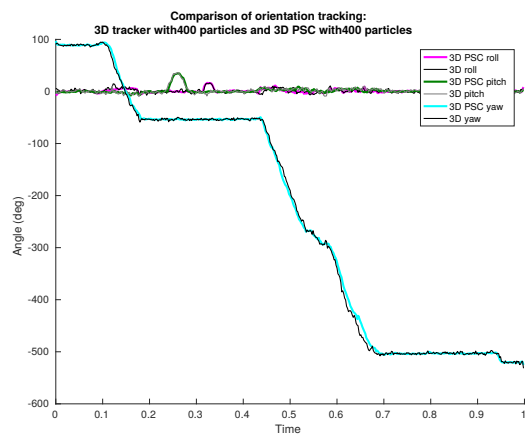
(c) Trajectory for translation (200 particles)



(d) Trajectory for orientation (200 particles)

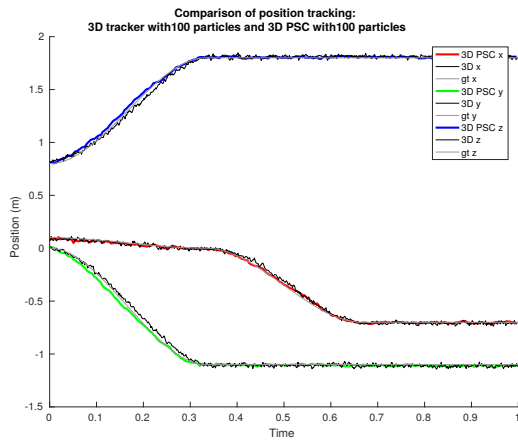


(e) Trajectory for translation (400 particles)

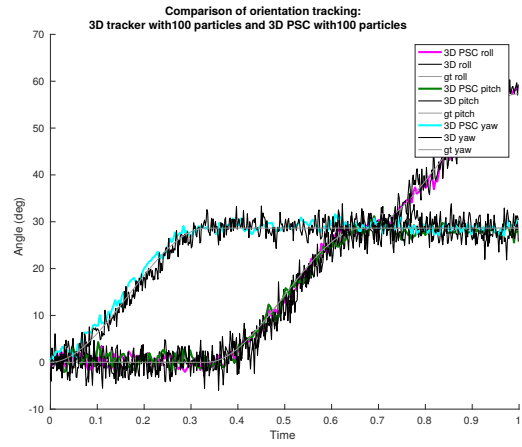


(f) Trajectory for orientation (400 particles)

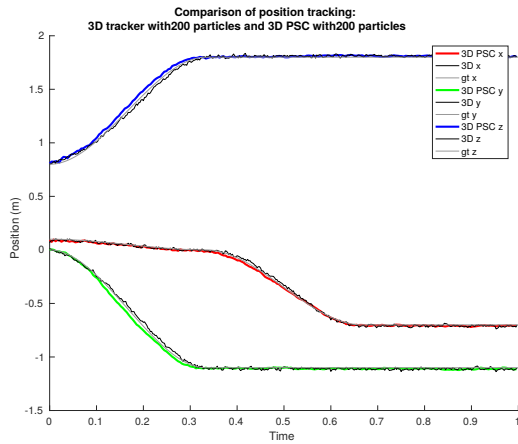
Figure B.1: Comparison of trajectories for object WoodBox using our live experimental setup



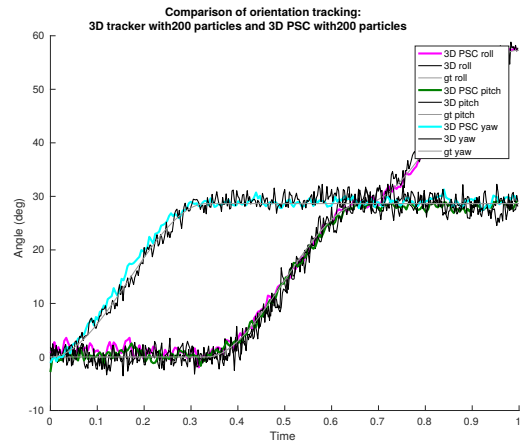
(a) Trajectory for translation (100 particles)



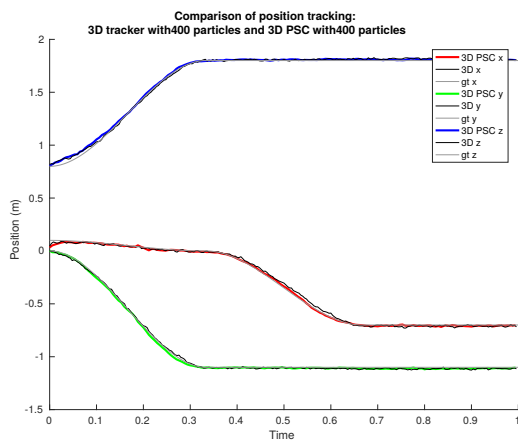
(b) Trajectory for orientation (100 particles)



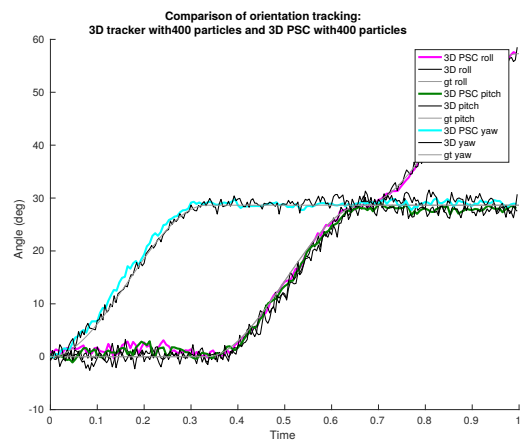
(c) Trajectory for translation (200 particles)



(d) Trajectory for orientation (200 particles)



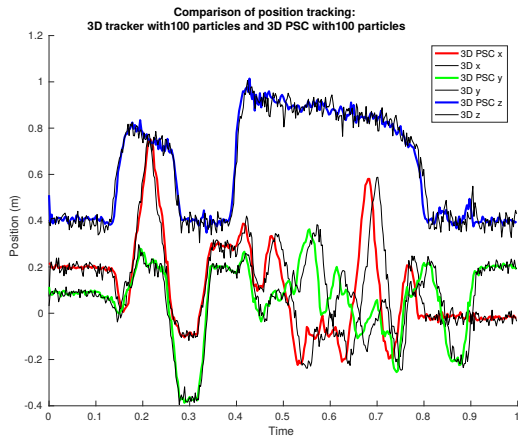
(e) Trajectory for translation (400 particles)



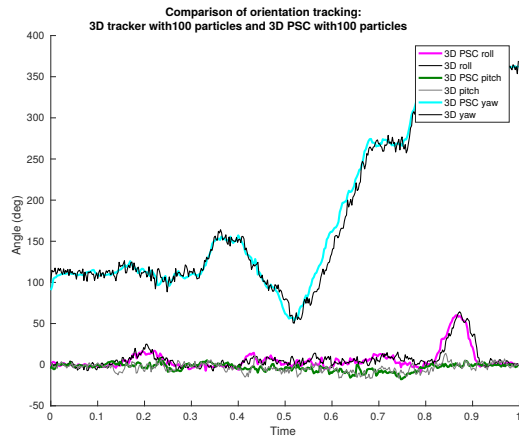
(f) Trajectory for orientation (400 particles)

Figure B.2: Comparison of trajectories for object WoodBox over synthetic data with ground truth

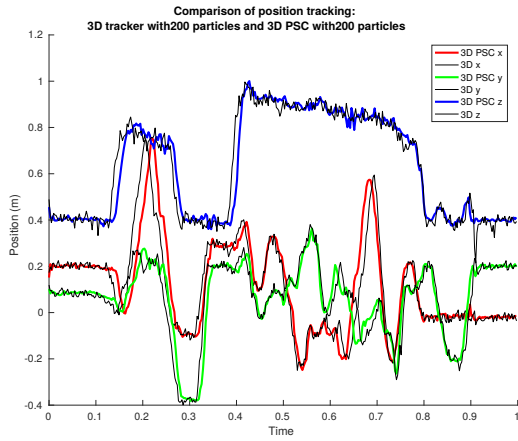
B. EVALUATION OF TRACKING WITH PRIMITIVE SHAPE CONSTRAINTS



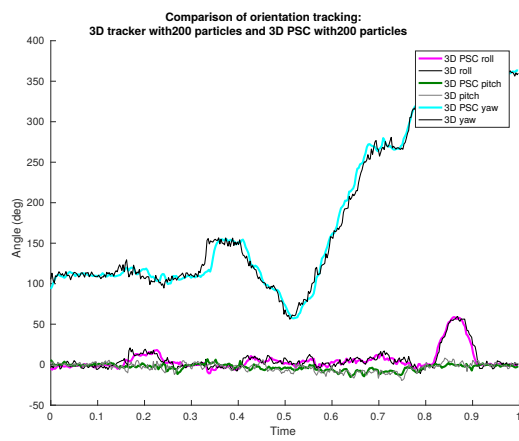
(a) Trajectory for translation (100 particles)



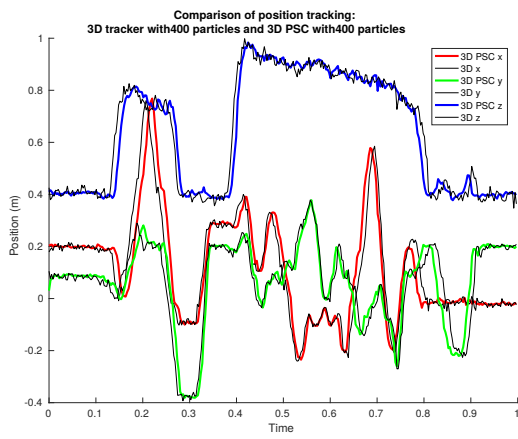
(b) Trajectory for orientation (100 particles)



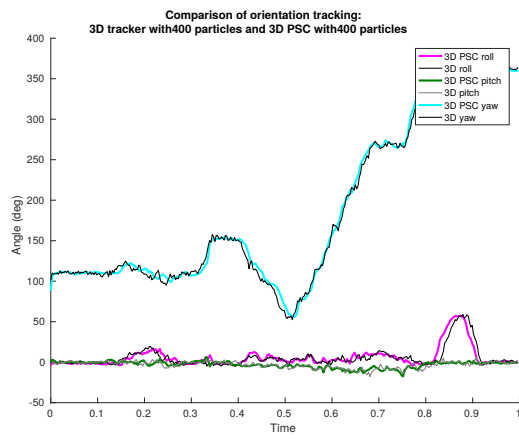
(c) Trajectory for translation (200 particles)



(d) Trajectory for orientation (200 particles)

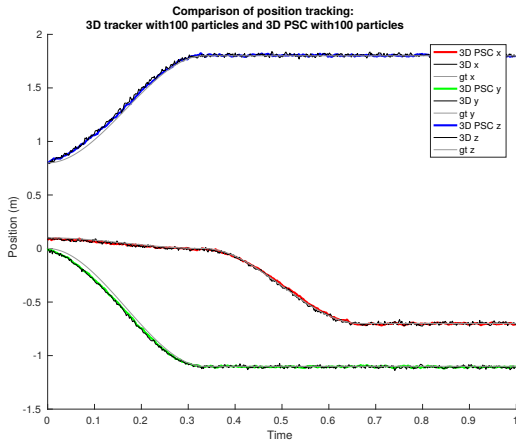


(e) Trajectory for translation (400 particles)

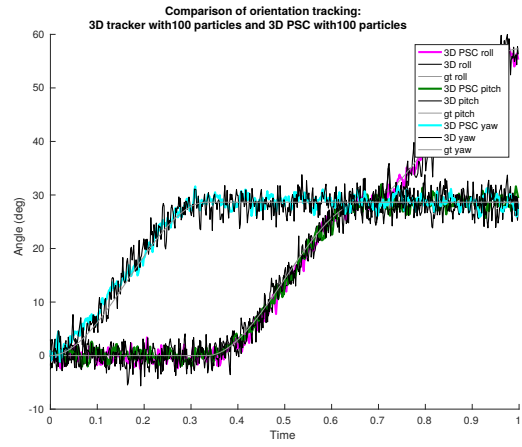


(f) Trajectory for orientation (400 particles)

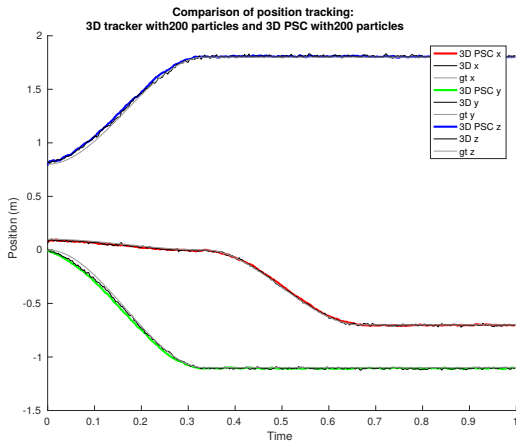
Figure B.3: Comparison of trajectories for object Box using our live experimental setup



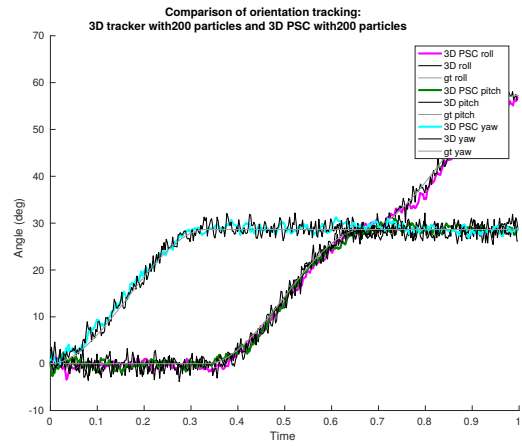
(a) Trajectory for translation (100 particles)



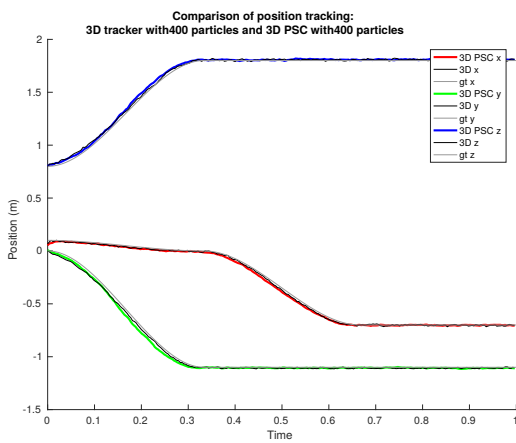
(b) Trajectory for orientation (100 particles)



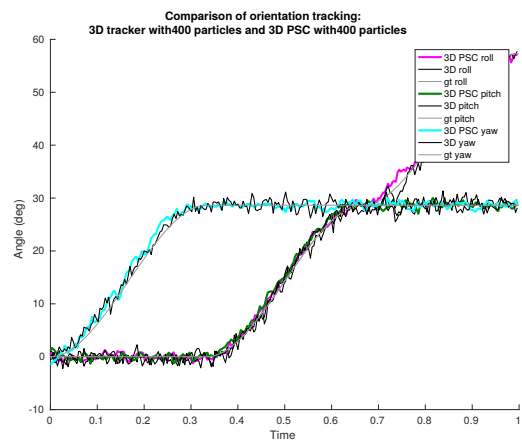
(c) Trajectory for translation (200 particles)



(d) Trajectory for orientation (200 particles)



(e) Trajectory for translation (400 particles)



(f) Trajectory for orientation (400 particles)

Figure B.4: Comparison of trajectories for object Box over synthetic data with ground truth

References

- [1] T. HODAN, P. HALUZA, Š. OBDRŽÁLEK, J. MATAS, M. LOURAKIS, AND X. ZABULIS. **T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-Less Objects**. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888, March 2017. 4
- [2] ALEXANDER PERZYLO, NIKHIL SOMANI, MARKUS RICKERT, AND ALOIS KNOLL. **An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015. 7, 24, 53, 104, 110
- [3] ALEXANDER PERZYLO, NIKHIL SOMANI, STEFAN PROFANTER, ANDRE GASCHLER, SASCHA GRIFFITHS, MARKUS RICKERT, AND ALOIS KNOLL. **Ubiquitous Semantics: Representing and Exploiting Knowledge, Geometry, and Language for Cognitive Robot Systems**. In *Proceedings of the Workshop Towards Intelligent Social Robots - Current Advances in Cognitive Robotics, IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*, Seoul, South Korea, November 2015. 7
- [4] ALEXANDER PERZYLO, NIKHIL SOMANI, STEFAN PROFANTER, INGMAR KESSLER, MARKUS RICKERT, AND ALOIS KNOLL. **Intuitive Instruction of Industrial Robots: Semantic Process Descriptions for Small Lot Production**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Republic of Korea, October 2016. <https://youtu.be/bbInEMEF5zU>. 7, 53, 68, 79
- [5] ALEXANDER PERZYLO, NIKHIL SOMANI, STEFAN PROFANTER, MARKUS RICKERT, AND ALOIS KNOLL. **Multimodal binding of parameters for task-based robot programming based on semantic descriptions of modalities and parameter types**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Multimodal Semantics for Robotic Systems*, Hamburg, Germany, September 2015. 7
- [6] NIKHIL SOMANI, MARKUS RICKERT, AND ALOIS KNOLL. **An exact solver for geometric constraints with inequalities**. *IEEE Robotics and Automation Letters*, **2**(2):1148–1155, April 2017. accepted for presentation at ICRA 2017. 12, 29, 54, 80, 102
- [7] NIKHIL SOMANI, MARKUS RICKERT, ANDRE GASCHLER, CAIXIA CAI, ALEXANDER PERZYLO, AND ALOIS KNOLL. **Task level robot programming using prioritized non-linear inequality constraints**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*,

REFERENCES

- Daejeon, Republic of Korea, October 2016. <https://youtu.be/baet9IkTK04>, <https://github.com/nsomani/constraint-controller-library>. 12, 29, 54, 74, 80, 102
- [8] NIKHIL SOMANI, CAIXIA CAI, ALEXANDER PERZYLO, MARKUS RICKERT, AND ALOIS KNOLL. **Object Recognition using constraints from Primitive Shape Matching**. In *Proceedings of the International Symposium on Visual Computing (ISVC)*, pages 783–792. Springer, December 2014. 12, 109
- [9] NIKHIL SOMANI, ALEXANDER PERZYLO, CAIXIA CAI, MARKUS RICKERT, AND ALOIS KNOLL. **Object detection using boundary representations of primitive shapes**. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Zhuhai, China, December 2015. 12, 109, 111
- [10] NIKHIL SOMANI, YAADHAV RAAJ, SURAJ NAIR, AND ALOIS KNOLL. **Adapting the Search Subspace of a Particle Filter using Geometric Constraints**. Technical Report TUM-I1762, 2017. 12, 119
- [11] A RODRIGUEZ, L BASAEZ, AND E CELAYA. **A Relational Positioning Methodology for Robot Task Specification and Execution**. *IEEE Transactions on Robotics*, **24**(3):600–611, jun 2008. 18, 20, 33, 53, 54, 55, 127
- [12] ELMER G. GILBERT, DANIEL W. JOHNSON, AND S. SATHIYA KEERTHI. **A fast procedure for computing the distance between complex objects in three-dimensional space**. *IEEE Journal of Robotics and Automation*, **4**(2):193–203, 1988. 25
- [13] JOHN SCHULMAN, JONATHAN HO, ALEX LEE, IBRAHIM AWWAL, HENRY BRADLOW, AND PIETER ABBEEL. **Finding locally optimal, collision-free trajectories with sequential convex optimization**. *Proceedings of Robotics: Science and Systems*, 2013. 25, 53
- [14] ANDRE GASCHLER, QUIRIN FISCHER, AND ALOIS KNOLL. **The Bounding Mesh Algorithm**. Technical Report TUM-I1522, Technische Universität München, Germany, June 2015. 25
- [15] ANDRE GASCHLER. *Efficient Geometric Predicates for Integrated Task and Motion Planning*. Dissertation, Technische Universität München. 25
- [16] T. YOSHIKAWA. **Manipulability of Robotic Mechanisms**. *The International Journal of Robotics Research*, **4**(2):3–9, 1985. 27, 53
- [17] BERNHARD BETTIG AND CHRISTOPH M. HOFFMANN. **Geometric Constraint Solving in Parametric Computer-Aided Design**. *Journal of Computing and Information Science in Engineering*, **11**(2):1–26, 2011. 29
- [18] CHRISTOPH M. HOFFMANN AND ROBERT JOAN-ARINYO. **A Brief on Constraint Solving**. *Computer-Aided Design and Applications*, **2**(5):655–663, jan 2005. 29
- [19] STEVEN G. JOHNSON. **The NLopt nonlinear-optimization package**. <http://ab-initio.mit.edu/nlopt> [cited 2018]. 38

-
- [20] M. J. D. POWELL. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994. 38
- [21] M. J. D. POWELL. **Direct search algorithms for optimization calculations**. *Acta Numerica*, **7**:287–336, 1998. 38
- [22] DIETER KRAFT. **Algorithm 733: TOMP–Fortran Modules for Optimal Control Calculations**. *ACM Trans. Math. Softw.*, **20**(3):262–281, Sep 1994. 38
- [23] D. KRAFT. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988. 38
- [24] STEFAN PROFANTER, ALEXANDER PERZYLO, NIKHIL SOMANI, MARKUS RICKERT, AND ALOIS KNOLL. **Analysis and Semantic Modeling of Modality Preferences in Industrial Human-Robot Interaction**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, sep 2015. 47, 53, 55, 79
- [25] MARTIN KRAFT AND MARKUS RICKERT. **How to Teach Your Robot in 5 Minutes: Applying UX Paradigms to Human-Robot-Interaction**. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Lisbon, Portugal, August 2017. 47, 53
- [26] 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):43–53, 1987. 52, 53
- [27] TOMAS LOZANO-PEREZ, JOSEPH L JONES, EMMANUEL MAZER, PATRICK O’DONNELL, ERIC WL GRIMSON, PIERRE TOURNASSOUD, ALAIN LANUSSE, ET AL. **Handey: A robot system that recognizes, plans, and manipulates**. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, **4**, pages 843–849, 1987. 52
- [28] O. KHATIB, L. SENTIS, J. PARK, AND J. WARREN. **Whole-Body Dynamic Behavior and Control of Human-Like Robots**. *International Journal of Humanoid Robotics*, **1**(1):29–43, 2004. 52, 54, 74, 80, 129
- [29] STEFANO CHIAVERINI, GIUSEPPE ORIOLO, AND IAN D. WALKER. **Kinematically Redundant Manipulators**. In BRUNO SICILIANO AND OUSSAMA KHATIB, editors, *Springer Handbook of Robotics*, pages 245–268. Springer Berlin Heidelberg, 2008. 52
- [30] A ESCANDE, N MANSARD, AND PB WIEBER. **Hierarchical quadratic programming: Fast online humanoid-robot motion generation**. *The International Journal of Robotics Research*, **33**(7):1006–1028, June 2014. 52, 53
- [31] JORIS DE SCHUTTER, TINNE DE LAET, JOHAN RUTGEERTS, WILM DECRÉ, RUBEN SMITS, ERWIN AERTBELIËN, KASPER CLAES, AND HERMAN BRUYNINCKX. **Constraint-based Task**

REFERENCES

- Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty.** *The International Journal of Robotics Research*, **26**(5):433–455, 2007. 52, 53, 54
- [32] GIANNI BORGHESAN AND ERWIN AERTBELI. **Constraint- and synergy-based specification of manipulation tasks.** *International Conference on Robotics and Automation*, 2014. 52
- [33] W. DECRE, R. SMITS, H. BRUYNINCKX, AND J. DE SCHUTTER. **Extending iTaSC to support inequality constraints and non-instantaneous task specification.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 964–971, May 2009. 52, 53, 54
- [34] E. AERTBELIEN AND J. DE SCHUTTER. **eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1540–1546, September 2014. 52
- [35] N. MANSARD, O. STASSE, P. EVRARD, AND A. KHEDDAR. **A versatile Generalized Inverted Kinematics implementation for collaborative working humanoid robots: The Stack Of Tasks.** In *International Conference on Advanced Robotics*, pages 1–6, June 2009. 52, 54
- [36] N. MANSARD, O. KHATIB, AND A. KHEDDAR. **A Unified Approach to Integrate Unilateral Constraints in the Stack of Tasks.** *IEEE Transactions on Robotics*, **25**(3):670–685, June 2009. 52, 54
- [37] ALEXANDER DIETRICH, CHRISTIAN OTT, AND ALIN ALBU-SCHÄFFER. **An overview of null space projections for redundant, torque-controlled robots.** *The International Journal of Robotics Research*, **34**(11):1385–1400, 2015. 52
- [38] R. LOBER, V. PADOIS, AND O. SIGAUD. **Multiple task optimization using dynamical movement primitives for whole-body reactive control.** In *IEEE-RAS International Conference on Humanoid Robots*, pages 193–198, Nov 2014. 52
- [39] ALEXANDER DIETRICH, CHRISTIAN OTT, AND ALIN ALBU-SCH. **Multi-Objective Compliance Control of Redundant Manipulators : Hierarchy , Control , and Stability.** pages 3043–3050, 2013. 52
- [40] ANDREA DEL PRETE, FRANCESCO NORI, GIORGIO METTA, AND LORENZO NATALE. **Prioritized Motion-Force Control of Constrained Fully-Actuated Robots: “Task Space Inverse Dynamics”.** *Robotics and Autonomous Systems*, **63**:150–157, jan 2015. 52, 53, 54, 74, 80, 129
- [41] CAIXIA CAI, NIKHIL SOMANI, MARKUS RICKERT, AND ALOIS KNOLL. **Prioritized Motion-Force Control of Multi-Constraints for Industrial Manipulators.** In *IEEE International Conference on Robotics and Biomimetics*, Zhuhai, China, December 2015. 52, 74, 80, 129
- [42] INGO KRESSE. *A semantic constraint-based Robot Motion Control for Generalizing Everyday Manipulation Actions.* Dissertation, Technische Universität München, München, 2017. 52, 53

-
- [43] G. BARTELS, I. KRESSE, AND M. BEETZ. **Constraint-based movement representation grounded in geometric features.** In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 547–554, Oct 2013. 52, 53
- [44] A. DIETRICH, T. WIMBOCK, A. ALBU-SCHAFFER, AND G. HIRZINGER. **Reactive Whole-Body Control: Dynamic Mobile Manipulation Using a Large Number of Actuated Degrees of Freedom.** *IEEE Robotics Automation Magazine*, **19**(2):20–33, June 2012. 52
- [45] FRANCESCO ROMANO, ANDREA DEL PRETE, NICOLAS MANSARD, FRANCESCO NORI, ANDREA DEL PRETE, NICOLAS MANSARD, FRANCESCO NORI, FRANCESCO ROMANO, ANDREA DEL PRETE, NICOLAS MANSARD, FRANCESCO NORI, PRIORITIZED OPTIMAL, FRANCESCO ROMANO, ANDREA DEL PRETE, NICOLAS MANSARD, AND FRANCESCO NORI. **Prioritized Optimal Control: a Hierarchical Differential Dynamic Programming approach.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4–10, 2015. 52
- [46] F CHAUMETTE AND E MARCHAND. **A Redundancy-Based Iterative Approach for Avoiding Joint Limits: Application to Visual Servoing.** *IEEE Transactions on Robotics and Automation*, **17**(5):719–730, October 2001. 53
- [47] OLIVIER STASSE, ADRIEN ESCANDE, NICOLAS MANSARD, SYLVAIN MIOSSEC, PAUL EVRARD, AND ABDERRAHMANE KHEDDAR. **Real-Time (Self) -Collision Avoidance Task on a HRP-2 Humanoid Robot.** *IEEE International Conference on Robotics and Automation*, **25**(3):3200–3205, 2008. 53
- [48] CLAUS LENZ, MARKUS RICKERT, GIORGIO PANIN, AND ALOIS KNOLL. **Constraint task-based control in industrial settings.** *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3058–3063, 2009. 53, 54
- [49] YOUNG WHEE SUNG, DONG KWON CHO, AND MYUNG JIN CHUNG. **A constrained optimization approach to resolving manipulator redundancy.** *Journal of Robotic Systems*, **13**(5):275–288, 1996. 53
- [50] T. DE LAET, S. BELLENS, R. SMITS, E. AERTBELIEN, H. BRUYNINCKX, AND J. DE SCHUTTER. **Geometric Relations Between Rigid Bodies (Part 1): Semantics for Standardization.** *IEEE Robotics Automation Magazine*, **20**(1):84–93, March 2013. 53
- [51] T DE LAET, S BELLENS, H BRUYNINCKX, AND J DE SCHUTTER. **Geometric Relations Between Rigid Bodies (Part 2): From Semantics to Software.** *IEEE Robotics Automation Magazine*, **20**(2):91–102, June 2013. 53
- [52] H. F. DURRANT-WHYTE. **Uncertain geometry in robotics.** *IEEE Journal on Robotics and Automation*, **4**(1):23–31, Feb 1988. 53
- [53] G. BORGHEGAN, E. SCIONI, A. KHEDDAR, AND H. BRUYNINCKX. **Introducing Geometric Constraint Expressions Into Robot Constrained Motion Specification and Control.** *IEEE Robotics and Automation Letters*, **1**(2):1140–1147, July 2016. 53

REFERENCES

- [54] A. NAKAMURA, T. OGASAWARA, T. SUEHIRO, AND H. TSUKUNE. **Skill-based backprojection for fine motion planning.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, **2**, pages 526–533 vol.2, Nov 1996. 53
- [55] A. NAKAMURA, T. OGASAWARA, K. KITAGAKI, AND T. SUEHIRO. **Using robust and simplified geometric models in skill-based manipulation.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, **1**, pages 138–145 vol.1, 2001. 53
- [56] BERND FINKEMEYER, TORSTEN KRÖGER, AND FRIEDRICH M WAHL. **Executing assembly tasks specified by manipulation primitive nets.** *Advanced Robotics*, **19**(5):591–611, 2005. 53
- [57] TORSTEN KRÖGER, BERND FINKEMEYER, AND FRIEDRICH M. WAHL. **Manipulation Primitives — A Universal Interface between Sensor-Based Motion Control and Robot Programming.** In DANIEL SCHÜTZ AND FRIEDRICH M. WAHL, editors, *Robotic Systems for Handling and Assembly*, **67** of *Springer Tracts in Advanced Robotics*, pages 293–313. Springer, 2011. 53
- [58] TORSTEN KRÖGER. **Opening the door to new sensor-based robot applications—The Reflexes Motion Libraries.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4. IEEE, 2011. 53
- [59] MAJ STENMARK, JACEK MALEC, AND ANDREAS STOLT. *From High-Level Task Descriptions to Executable Robot Code*, pages 189–202. Springer International Publishing, Cham, 2015. 53
- [60] RASMUS HASLE ANDERSEN, THOMAS SOLUND, AND JOHN HALLAM. **Definition and Initial Case-Based Evaluation of Hardware-Independent Robot Skills for Industrial Robotic Co-Workers.** In *Proceedings of the International Symposium on Robotics*, pages 1–7, June 2014. 53
- [61] ULRIKE THOMAS, BERND FINKEMEYER, TORSTEN KROGER, AND FRIEDRICH M WAHL. **Error-tolerant execution of complex robot tasks based on skill primitives.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, **3**, pages 3069–3075. IEEE, 2003. 53
- [62] FRANCESCO ROVIDA, MATTHEW CROSBY, DIRK HOLZ, ATHANASIOS S. POLYDOROS, BJARNE GROSSMANN, RONALD P. A. PETRICK, AND VOLKER KRÜGER. *SkiROS—A Skill-Based Robot Control Platform on Top of ROS*, pages 121–160. Springer International Publishing, Cham, 2017. 53
- [63] MAJ STENMARK AND JACEK MALEC. **Describing constraint-based assembly tasks in unstructured natural language.** *IFAC Proceedings Volumes*, **47**(3):3056 – 3061, 2014. 19th IFAC World Congress. 53
- [64] NATHAN RATLIFF, MATT ZUCKER, J ANDREW BAGNELL, AND SIDDHARTHA SRINIVASA. **CHOMP: Gradient optimization techniques for efficient motion planning.** In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 489–494, 2009. 53
- [65] NIKHIL SOMANI, ANDRE GASCHLER, MARKUS RICKERT, ALEXANDER PERZYLO, AND ALOIS KNOLL. **Constraint-Based Task Programming with CAD Semantics: From Intuitive**

- Specification to Real-Time Control.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, September 2015. <https://youtu.be/qRJ1JmNoFEw>. 54, 74, 80, 104
- [66] ALEXANDER PERZYLO, NIKHIL SOMANI, STEFAN PROFANTER, MARKUS RICKERT, AND ALOIS KNOLL. **Toward Efficient Robot Teach-in and Semantic Process Descriptions for Small Lot Sizes.** In *Proceedings of Robotics: Science and Systems, Workshop on Combining AI Reasoning and Cognitive Science with Robotics*, Rome, Italy, July 2015. <http://youtu.be/B1Qu8Mt3WtQ>. 55, 92
- [67] MATHIAS HAAGE, STEFAN PROFANTER, INGMAR KESSLER, ALEXANDER PERZYLO, NIKHIL SOMANI, OLOF SÖRNMO, MARTIN KARLSSON, SVEN GESTEGÅRD ROBERTZ, KLAS NILSSON, LUDOVIC RESCH, AND MICHAEL MARTI. **On Cognitive Robot Woodworking in SMERobotics.** In *International Symposium on Robotics (ISR)*, Munich, Germany, June 2016. 64
- [68] MARKUS RICKERT. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems.* Dissertation, Technische Universität München, 2011. 68
- [69] MARKUS RICKERT AND ANDRE GASCHLER. **Robotics Library: An Object-Oriented Approach to Robot Applications.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, September 2017. 68, 73, 75
- [70] STEVEN M. LAVALLE. **Rapidly-Exploring Random Trees: A New Tool for Path Planning.** Technical report, 1998. 79
- [71] J. J. KUFFNER AND S. M. LAVALLE. **RRT-connect: An efficient approach to single-query path planning.** In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, **2**, pages 995–1001 vol.2, 2000. 79
- [72] MARKUS RICKERT, ARNE SIEVERLING, AND OLIVER BROCK. **Balancing Exploration and Exploitation in Sampling-Based Motion Planning.** *IEEE Transactions on Robotics*, **30**(6):1305–1317, December 2014. 79, 81
- [73] STEVEN M. LAVALLE, JAMES J. KUFFNER, AND JR. **Rapidly-Exploring Random Trees: Progress and Prospects.** In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2000. 80
- [74] YOSHIHITO KOGA, KOICHI KONDO, JAMES KUFFNER, AND JEAN-CLAUDE LATOMBE. **Planning Motions with Intentions.** In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94*, pages 395–408, New York, NY, USA, 1994. ACM. 80, 81
- [75] KATSU YAMANE, JAMES J. KUFFNER, AND JESSICA K. HODGINS. **Synthesizing Animations of Human Manipulation Tasks.** In *ACM SIGGRAPH 2004 Papers, SIGGRAPH '04*, pages 532–539, New York, NY, USA, 2004. ACM. 80, 81

REFERENCES

- [76] ZHENWANG YAO AND K. GUPTA. **Path planning with general end-effector constraints: using task space to guide configuration space search.** In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1875–1880, Aug 2005. 80, 81
- [77] D. BERTRAM, J. KUFFNER, R. DILLMANN, AND T. ASFOUR. **An integrated approach to inverse kinematics and path planning for redundant manipulators.** In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1874–1879, May 2006. 80, 81
- [78] E. DRUMWRIGHT AND V. NG-THOW-HING. **Toward Interactive Reaching in Static Environments for Humanoid Robots.** In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 846–851, Oct 2006. 80, 81
- [79] M. STILMAN. **Task constrained motion planning in robot joint space.** pages 3074–3081, Oct 2007. 80, 81
- [80] DMITRY BERENSON, SIDDHARTHA SRINIVASA, AND JAMES KUFFNER. **Task Space Regions: A Framework for Pose-constrained Manipulation Planning.** *International Journal of Robotic Research*, **30**(12):1435–1460, October 2011. 80, 81, 82
- [81] YU ZHONG. **Intrinsic shape signatures: A shape descriptor for 3D object recognition.** In *Computer Vision Workshops (ICCV Workshops), 2009*, pages 689–696, 2009. 92, 109
- [82] ANDREW E. JOHNSON AND MARTIAL HEBERT. **Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes.** *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**(5):433–449, May 1999. 92
- [83] R. B. RUSU, N. BLODOW, AND M. BEETZ. **Fast Point Feature Histograms (FPFH) for 3D registration.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, May 2009. 92
- [84] FEDERICO TOMBARI AND LUIGI DI STEFANO. *Interest Points via Maximal Self-Dissimilarities*, pages 586–600. Springer International Publishing, Cham, 2015. 92
- [85] F. TOMBARI, A. FRANCHI, AND L. DI. **BOLD Features to Detect Texture-less Objects.** In *IEEE International Conference on Computer Vision*, pages 1265–1272, Dec 2013. 92
- [86] SAMUELE SALTI, FEDERICO TOMBARI, AND LUIGI DI STEFANO. *On the Use of Implicit Shape Models for Recognition of Object Categories in 3D Data*, pages 653–666. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. 92
- [87] FEDERICO TOMBARI, SAMUELE SALTI, AND LUIGI DI STEFANO. **Unique Shape Context for 3D Data Description.** In *Proceedings of the ACM Workshop on 3D Object Retrieval, 3DOR '10*, pages 57–62, New York, NY, USA, 2010. ACM. 92
- [88] RADU BOGDAN RUSU, GARY BRADSKI, ROMAIN THIBAU, AND JOHN HSU. **Fast 3d recognition and pose using the viewpoint feature histogram.** In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2155–2162. IEEE, 2010. 92, 109

-
- [89] NABILA ZRIRA, FATIMA ZAHRA OUADIAY, MOHAMED HANNAT, EL HOUSSINE BOUYAKHF, AND MOHAMED MAJID HIMMI. **Evaluation of PCL's Descriptors for 3D Object Recognition in Cluttered Scene**. In *Proceedings of the 2nd International Conference on Big Data, Cloud and Applications*, BDCA'17, pages 81:1–81:6, New York, NY, USA, 2017. ACM. 92
- [90] NIKHIL SOMANI, EMMANUEL DEAN-LEON, CAIXIA CAI, AND ALOIS KNOLL. **Scene Perception and Recognition in industrial environments**. In *9th International Symposium on Visual Computing (ISVC)*. Springer, July 2013. 92, 100, 103, 109
- [91] CHAVDAR PAPAZOV, SAMI HADDADIN, SVEN PARUSEL, KAI KRIEGER, AND DARIUS BURSCHKA. **Rigid 3D geometry matching for grasping of known objects in cluttered scenes**. *International Journal of Robotic Research*, **31**:538–553, 2012. 92
- [92] CHAVDAR PAPAZOV AND DARIUS BURSCHKA. **An efficient RANSAC for 3D object recognition in noisy and occluded scenes**. *Proceedings of the 10th Asian conference on Computer vision - Volume Part I*, pages 135–148, 2011. 92, 109
- [93] ULRIKE THOMAS. **Stable Pose Estimation Using Ransac with Triple Point Feature Hash Maps and Symmetry Exploration**. In *Machine Vision Applications*, pages 109–112, 2013. 92, 109
- [94] RUWEN SCHNABEL, RAOUL WESSEL, ROLAND WAHL, AND REINHARD KLEIN. **Shape Recognition in 3D Point-Clouds**. In V. SKALA, editor, *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*. UNION Agency-Science Press, feb 2008. 92, 109
- [95] R. SCHNABEL, R. WAHL, AND R. KLEIN. **Efficient RANSAC for Point-Cloud Shape Detection**. *Computer Graphics Forum*, **26**(2):214–226, 2007. 92
- [96] RUWEN SCHNABEL. *Efficient Point-Cloud Processing with Primitive Shapes*. Dissertation, Universität Bonn, December 2010. 92
- [97] MATTHIAS NIEUWENHUISEN, JOERG STÜCKLER, ALEXANDER BERNER, REINHARD KLEIN, AND SVEN BEHNKE. **Shape-Primitive Based Object Recognition and Grasping**. In *7th German Conference on Robotics (ROBOTIK)*, May 2012. 92
- [98] ALEXANDER BERNER, JUN LI, DIRK HOLZ, JOERG STÜCKLER, SVEN BEHNKE, AND REINHARD KLEIN. **Combining Contour and Shape Primitives for Object Detection and Pose Estimation of Prefabricated Parts**. In *IEEE International Conference on Image Processing (ICIP), Melbourne, Australia*, September 2013. 92
- [99] YURI BOYKOV, OLGA VEKSLER, AND RAMIN ZABIH. **Fast Approximate Energy Minimization via Graph Cuts**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **23**(11):1222–1239, November 2001. 96
- [100] Y. BOYKOV AND V. KOLMOGOROV. **An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**(9):1124–1137, 2004. 96

REFERENCES

- [101] A. DELONG, A. OSOKIN, H.N. ISACK, AND Y. BOYKOV. **Fast approximate energy minimization with label costs**. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2173–2180, 2010. 96
- [102] ANDREW DELONG, ANTON OSOKIN, HOSSAM N. ISACK, AND YURI BOYKOV. **Fast Approximate Energy Minimization with Label Costs**. *International Journal on Computer Vision*, **96**(1):1–27, January 2012. 96
- [103] A. RICHTSFELD, T. MORWALD, J. PRANKL, M. ZILlich, AND M. VINCZE. **Segmentation of unknown objects in indoor environments**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4791–4796, 2012. 100
- [104] ALPER YILMAZ, OMAR JAVED, AND MUBARAK SHAH. **Object Tracking: A Survey**. *ACM Computing Surveys*, **38**(4), December 2006. 107
- [105] ZHE CHEN. **Bayesian filtering: From Kalman filters to particle filters, and beyond**. *Statistics*, **182**(1):1–69, 2003. 108
- [106] YAAKOV BAR-SHALOM. *Tracking and data association*. Academic Press Professional, Inc., 1987. 108
- [107] SURAJ NAIR, GIORGIO PANIN, MARTIN WOJTCZYK, CLAUS LENZ, THOMAS FRIEDELHUBER, AND ALOIS KNOLL. **A Multi-Camera Person Tracking System for Robotic Applications in Virtual Reality TV Studio**. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2008. 108
- [108] SURAJ NAIR, GIORGIO PANIN, THORSTEN RÖDER, THOMAS FRIEDELHUBER, AND ALOIS KNOLL. **A Distributed and Scalable Person Tracking System for Robotic Visual Servoing with 8 DOF in Virtual Reality TV Studio Automation**. In *Proceedings of the 6th International Symposium on Mechatronics and its Applications (ISMA09)*. IEEE, 2009. 108
- [109] CHANGHYUN CHOI AND HENRIK I. CHRISTENSEN. **RGB-D object tracking: A particle filter approach on GPU**. *IEEE International Conference on Intelligent Robots and Systems*, pages 1084–1091, 2013. 108, 117
- [110] GIORGIO PANIN, CLAUS LENZ, MARTIN WOJTCZYK, SURAJ NAIR, ERWIN ROTH, THOMAS FRIEDELHUBER, AND ALOIS KNOLL. **A unifying software architecture for model-based visual tracking**, 2008. 109
- [111] STEFAN HINTERSTOISSER, VINCENT LEPETIT, SLOBODAN ILIC, STEFAN HOLZER, GARY BRADSKI, KURT KONOLIGE, AND NASSIR NAVAB. **Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes**. In *Asian conference on computer vision*, pages 548–562. Springer, 2012. 109
- [112] STEFAN HINTERSTOISSER, STEFAN HOLZER, CEDRIC CAGNIART, SLOBODAN ILIC, KURT KONOLIGE, NASSIR NAVAB, AND VINCENT LEPETIT. **Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes**. In *International Conference on Computer Vision*, pages 858–865. IEEE, 2011. 109

-
- [113] JAMES ANTHONY BROWN AND DAVID W CAPSON. **A framework for 3D model-based visual tracking using a GPU-accelerated particle filter.** *IEEE Transactions on Visualization and Computer Graphics*, **18**(1):68–80, 2012. 109, 110
- [114] STAN MELAX, LEONID KESELMAN, AND STERLING ORSTEN. **Dynamics Based 3D Skeletal Hand Tracking.** In *Proceedings of Graphics Interface 2013*, GI '13, pages 63–70, Toronto, Ont., Canada, Canada, 2013. Canadian Information Processing Society. 109
- [115] CHEN QIAN, XIAO SUN, YICHEN WEI, XIAOOU TANG, AND JIAN SUN. **Realtime and robust hand tracking from depth.** In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1106–1113. IEEE, 2014. 109
- [116] IASON OIKONOMIDIS, NIKOLAOS KYRIAZIS, AND ANTONIS A ARGYROS. **Efficient model-based 3D tracking of hand articulations using Kinect.** In *British Machine Vision Conference (BMVC)*, **1**, page 3, 2011. 109
- [117] HENNING HAMER, KONRAD SCHINDLER, ESTHER KOLLER-MEIER, AND LUC VAN GOOL. **Tracking a hand manipulating an object.** In *Computer Vision, 2009 IEEE 12th International Conference On*, pages 1475–1482. IEEE, 2009. 109
- [118] BRANKO RISTIC, SANJEEV ARULAMPALAM, AND NEIL GORDON. *Beyond the Kalman filter: Particle filters for tracking applications*, **685**. Artech house Boston, 2004. 109
- [119] JAN GIEBEL, DARIN M GAVRILA, AND CHRISTOPH SCHNÖRR. **A bayesian framework for multi-cue 3d object tracking.** In *European Conference on Computer Vision*, pages 241–252. Springer, 2004. 109
- [120] DIETER FOX. **Adapting the Sample Size in Particle Filters Through KLD-Sampling.** *The International Journal of Robotics Research*, **22**(12):985–1003, 2003. 110
- [121] G. PANIN. *Model-based Visual Tracking: The OpenTL Framework*. IT Pro. Wiley, 2011. 110
- [122] LICONG ZHANG, J STURM, D CREMERS, AND DONGHEUI LEE. **Real-time human motion tracking using multiple depth cameras.** *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2389–2395, 2012. 110
- [123] JENS BEHLEY, VOLKER STEINHAGE, AND ARMIN B. CREMERS. **Efficient Radius Neighbor Search in Three-dimensional Point Clouds.** In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015. 110
- [124] GREG PASS AND RAMIN ZABIH. **Comparing Images Using Joint Histograms.** *Multimedia Systems*, **7**(3):234–240, May 1999. 115
- [125] ANIL BHATTACHARYYA. **On a measure of divergence between two multinomial populations.** *Sankhyā: the indian journal of statistics*, pages 401–406, 1946. 115
- [126] M. ISARD AND A. BLAKE. **Condensation – conditional density propagation for visual tracking.** *International Journal of Computer Vision (IJCV)*, **29**(1):5–28, 1998. 115

REFERENCES

- [127] THIEMO WIEDEMEYER. **IAI Kinect2**. https://github.com/code-iai/iai_kinect2, 2014. Accessed June 12, 2015. 116
- [128] MARC ALEXA, JOHANNES BEHR, DANIEL COHEN-OR, SHACHAR FLEISHMAN, DAVID LEVIN, AND CLAUDIO T. SILVA. **Computing and rendering point set surfaces**. *IEEE Transactions on Visualization and Computer Graphics*, **9**(1):3–15, 2003. 117
- [129] IASON OIKONOMIDIS, NIKOLAOS KYRIAZIS, AND ANTONIS A ARGYROS. **Efficient model-based 3D tracking of hand articulations using Kinect**. In *BMVC*, **1**, page 3, 2011. 129
- [130] KOEN BUYS, CEDRIC CAGNIART, ANATOLY BAKSHEEV, TINNE DE LAET, JORIS DE SCHUTTER, AND CAROLINE PANTOFARU. **An Adaptable System for RGB-D Based Human Body Detection and Pose Estimation**. *J. Vis. Comun. Image Represent.*, **25**(1):39–52, January 2014. 129
- [131] CAIXIA CAI. *6D Visual Servoing for Industrial Manipulators applied to Human-Robot Interaction Scenarios*. Dissertation (phd thesis), Technische Universität München, Munich, Germany, 2017. 129
- [132] CAIXIA CAI, NIKHIL SOMANI, AND ALOIS KNOLL. **Orthogonal Image Features for Visual Servoing of a 6-DOF Manipulator with Uncalibrated Stereo Cameras**. *IEEE transactions on Robotics*, April 2016. 129
- [133] CAIXIA CAI, EMMANUEL DEAN-LEON, NIKHIL SOMANI, AND ALOIS KNOLL. **6D Image-based Visual Servoing for Robot Manipulators with Uncalibrated Stereo Cameras**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2014. 129
- [134] CAIXIA CAI, EMMANUEL DEAN-LEON, DARIO MENDOZA, NIKHIL SOMANI, AND ALOIS KNOLL. **Uncalibrated 3D Stereo Image-based Dynamic Visual Servoing for Robot Manipulators**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013. 129
- [135] D. J. AGRAVANTE, G. CLAUDIO, F. SPINDLER, AND F. CHAUMETTE. **Visual Servoing in an Optimization Framework for the Whole-Body Control of Humanoid Robots**. *IEEE Robotics and Automation Letters*, **2**(2):608–615, April 2017. 129