

TECHNISCHE UNIVERSITÄT MÜNCHEN

Institut für Informatik

Lehrstuhl für Bildverstehen und wissensbasierte Systeme

A semantic constraint-based Robot Motion Control for Generalizing Everyday Manipulation Actions

Ingo Kresse

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende: Univ.-Prof. Gudrun Klinker, Ph.D.

Prüfer der Dissertation: 1. Univ.-Prof. Michael Beetz, Ph.D.
Universität Bremen

2. Univ.-Prof. Dr. Alin Albu-Schäffer

Die Dissertation wurde am 23.08.2016 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13.12.2017 angenommen.

Abstract

Today, robots are entering human environments and must deal with clutter, uncertain sensor readings and open-ended tasks. Sometimes they are even competing directly with humans in terms of flexibility and reliability of their movements.

Humans can competently adapt their movements to various changes in their environment, like changed object locations, different object shapes, clutter or different furniture. They know implicitly what is important for a given movement and where there is freedom that can be exploited to adapt to these changes. Robots, on the other hand, are quite limited in their ability to adapt their movements when they encounter substantial changes in their environment: Only small variations in object poses can be compensated.

This thesis attempts to endow robots with some of the task-knowledge that makes humans act so competently in their every-day activities. Rather than exploiting the task-inherent freedoms to build challenging multi-tasking scenarios, the focus is on simple every-day tasks and their flexible, robust and autonomous execution, accommodating as many changes as possible.

A constraint-based approach is used to construct movements of every-day manipulation tasks. This representation is shared on a detailed, degree-of-freedom level with the whole robot architecture: The perception contributes detailed object knowledge in the form of geometric features like edge, corners, planes or symmetry axes, the high-level planner defines constraints which relate these features to each other in terms of both position and force, and the movement execution automatically translates these constraint descriptions into a controller setup for an impedance controlled robot.

This thesis introduces some basic reasoning tools for such constraints which can check whether two constraints are redundant or whether a constraint is well-defined in a given situation. It outlines how constraints can be derived from natural-language source like web instructions, from observed human movement data or from simulation, given some desired and undesired effects. It describes several ways to exploit the freedom, the so-constructed null space of the tasks. Finally, it examines the specification of force constraints in a contact-formation task, taking into account the precision limits when implemented on an impedance-controlled robot.

These approaches were evaluated in simulation and reality on two state-of-the-art robot platforms and several tasks in a kitchen scenario.

Kurzfassung

Heutzutage werden Roboter zunehmend in menschlichen Umgebungen eingesetzt und müssen mit Hindernissen, unsicheren Sensormessungen und offenen Aufgabenstellungen umgehen. Manchmal stehen sie gar in direktem Wettbewerb zu Menschen, wenn es um flexible und zuverlässige Bewegungen geht.

Menschen können ihre Bewegungen sehr kompetent an sehr verschiedene Änderungen ihrer Umgebung anpassen, wie etwa geänderte Positionen von Objekten, andere Formen von Objekten, Hindernisse oder andere Möbel. Menschen wissen implizit was an einer Bewegung wichtig ist und wo es Freiheiten gibt, die sie ausnutzen können um Änderungen der Umgebung gerecht zu werden.

Roboter sind dagegen recht beschränkt in ihrer Fähigkeit, Bewegungen anzupassen wenn sich ihre Umgebung erheblich ändert: Lediglich kleine Abweichungen von Objekt-Posen können sie kompensieren.

Diese Arbeit versucht, Roboter mit einem Teil des Aufgaben-Wissens zu versehen, die es den Menschen erlaubt, ihre alltäglichen Bewegungen so kompetent auszuführen. Anstatt die Freiheiten, die diesen Aufgaben innewohnen auszunutzen um anspruchsvolle Multi-Tasking-Szenarien zu lösen, liegt der Fokus auf einfachen, alltäglichen Aufgaben und ihrer flexiblen, robusten und autonomen Ausführung, die mit so vielen Änderungen der Umgebung wie möglich umgehen kann.

Es wird ein Constraint-basierter Ansatz verwendet um Bewegungen für alltägliche Aufgaben zu konstruieren. Diese detaillierte Repräsentation wird an die ganze Robotersteuerungs-Architektur verteilt: Die Perzeption erkennt geometrische Merkmale von Objekten wie Kanten, Ecken, Flächen oder Symmetrieachsen. Der High-Level-Planer erzeugt Constraints, die diese Merkmale in Beziehung zueinander setzen und legt sowohl Kräfte als auch Positionen fest. Die Bewegungsausführung übersetzt diese Con-

straints automatisch in eine Regler-Parametrierung für einen Impedanz-geregelten Roboter.

Diese Arbeit führt weiterhin einige grundlegende Werkzeuge ein um etwa zu überprüfen, ob zwei Constraints redundant sind oder ob ein Constraint in einer gegebenen Situation wohldefiniert ist.

Sie umreißt, wie solche Constraints automatisch erzeugt werden können, etwa aus natürlichsprachlichen Quellen wie Web-Anweisungen, aus menschlichen Bewegungsdaten oder aus Physik-Simulation in Verbindung mit erwünschten und unerwünschten Effekten. Mehrere Möglichkeiten werden beschrieben um den so erzeugten Nullraum auszunutzen. Schließlich wird die Spezifikation von Kraft-Constraints untersucht, die sicheren, definierten Kontakt zwischen zwei Objekten herstellen sollen. Dabei wird auch die begrenzte Genauigkeit eines Impedanz-geregelten Roboters in Betracht gezogen.

Diese Ansätze wurden anhand mehrerer Aufgaben in einem Küchen-Szenario getestet, sowohl in der Simulation als auch in der Realität an zwei State-of-the-Art mobilen manipulierenden Plattformen.

Acknowledgements

I'd like to thank my advisor Michael Beetz for his guidance, for providing the hardware that was the basis of my work, his insights and his vision for a research problem that seems so easy but is actually very deep. Thank you for letting me find and pursue my topic in this exciting field of research.

Constructing robots was great fun which I shared with my colleagues of the Intelligent Autonomous Systems Lab, thank you all for the great time. Especially Lorenz Mösenlechner and my office mates Federico Ruiz, Alexis Maldonado and Georg Bartels with whom I shared various discussions about robots, life and all. Thank you Georg for continuing my work and thank you Alexis, for your admirable ability to always see the good side of things. Without Ulrich Klank, the robots wouldn't have the perception that was needed for my work – or most of the demos we showed to the public. Thanks goes also to the students who helped making the robots run, especially Jonathan Kleinhellefort, Uwe Hermann, Julian Brunner and Humberto Alvarez.

I also want express my gratitude to the researchers from KU Leuven, especially Ruben Smits, who wrote KDL and introduced me to his iTaSC implementation on which I could base my work. Thank you, Tinne deLaet for inviting me as a guest in your house and thanks, Joris deSchutter and Herman Bruyninckx for encouraging me to continue this line of research that ultimately lead to this thesis.

Thanks go to my parents and my wife who has encouraged me to do a PhD in the first place. A very special thanks goes to my parents in law and my wife who made it possible for me to finish writing.

This thesis was partly funded through the Cotesys (Cognition in technical Systems) cluster of excellence and profited greatly from ROS (Quigley et al., 2009), especially from rviz.

Contents

Abstract	III
Kurzfassung	V
Acknowledgements	VII
Contents	IX
List of Resources	XIII
1. Introduction	1
2. Conceptual Framework	9
2.1. Definition of the Movement Specification Language	12
2.1.1. Geometric Features	13
2.1.2. Feature Functions	15
2.1.3. Desired Ranges	17
2.1.4. Execution	18
2.1.5. Integration into Robot Software Architecture	19
2.2. Related Concepts	22
2.2.1. Related Movement Representations	22
2.2.2. Geometric Features	25
2.2.3. Feature Functions	26
2.2.4. Constraint Ranges	29
2.3. Generating Movement Specifications	30
2.3.1. Perception of Geometric Features	30
2.3.2. Selection of Feature Functions	31
2.3.3. Constraint Ranges	34

2.3.4. Specification of Force Constraints	36
2.3.5. Reasoning Support	37
2.4. Executing Movements	40
2.4.1. Control Framework	40
2.4.2. Null Space Exploitation	43
2.5. Discussion	44
3. Generation of movement specifications	45
3.1. Geometric Feature Detection	46
3.1.1. Tool Classification	48
3.1.2. Feature Detection Scene Setup	49
3.1.3. Feature detection and localization	51
3.1.4. Tool Classification Results	54
3.1.5. CAD-Model Feature Extraction	55
3.2. Task Planning	59
3.2.1. Comparison to 'classical' planning task	61
3.2.2. Web instructions	62
3.2.3. Robot Knowledge Base and perception	64
3.2.4. Human observation	69
3.2.5. Parametrizing movements from desired effects through simula- tion	73
3.2.6. Monitoring	77
3.3. Planning Force-Controlled Movements	79
3.4. Examples for Constraint-based Task Descriptions	83
4. Execution of movements	87
4.1. Task Functions	88
4.1.1. Virtual Kinematic Chains	89
4.1.2. Feature-Functions	96
4.1.3. Visualization of Task Functions	100
4.1.4. Dependency Evaluation	105
4.1.5. End-effectors with limited Dexterity	110
4.1.6. Discontinuity Detection	110
4.1.7. Evaluation	113
4.1.8. Conclusion	116

4.2. Controller Architecture	117
4.2.1. Control Scheme	118
4.2.2. Range Controller	122
4.2.3. Null Space Exploitation	125
4.2.4. Joint Limit Avoidance	127
4.2.5. Manipulability Optimization	128
4.2.6. Optimize for next movement	129
4.2.7. Optimize for speed of next movement	129
4.2.8. Evaluation: Extension of Work Space	131
4.3. Implementation of Force Constraints	133
4.3.1. Relationship between Force/Torque Control and Impedance Control	133
4.3.2. Approximating Position/Force Control using Impedance Control	134
4.3.3. Deriving a desired Stiffness	136
4.3.4. Composing a Stiffness matrix from Feature Constraints	140
4.3.5. Recovering Coordinate System from Stiffness matrix	141
4.3.6. Evaluation	144
5. Conclusion	149
List of Prior Publications	153
A. Appendix	155
A.1. Twist Transformation	156
A.2. Inverse Twist Transformation	158
A.3. Stiffness Matrix Transformation	159
A.4. Twist Exponentials	160
A.5. Weighted Damped Least Squares Pseudo Inverse	162
A.6. Probabilistic Comparison of Functions	164
A.7. Task Function Comparison	165
Bibliography	169

List of Resources

Figures

1.1. The robot TUM-Rosie, flipping a pancake	1
1.2. Bridging of symbolic- and control engineering approaches	4
1.3. Detected line and plane features on a spatula	6
2.1. Pancake flipping task specified as constraints	10
2.2. Movement phases for the pancake flipping task	12
2.3. Geometric features that are used for control	14
2.4. Geometric features detected in kitchen tools	14
2.5. Feature Functions for the pancake flipping task	15
2.6. Constraints for 'above' relation	20
2.7. Constraints for 'above' relation, simplified	20
2.8. Overview of the system architecture	21
2.9. Dependent alignment constraints	38
2.10. Monitoring of a pancake flipping task	39
3.1. Detected line and plane features on a skimmer and a spatula	46
3.2. Frame placement on cylinder	47
3.3. The kitchen tools that were used for classification	48
3.4. Plane detection	51
3.5. Height maps of tools	52
3.6. Concavity Detection.	53
3.7. The line fitting process.	53
3.8. Examples for results of hole detection and edge extraction.	54
3.9. Visualization of several results (Holes, Edges)	55

3.10. The visual exploration results for the set of tools	56
3.11. Overview of the CAD model reasoning by (Tenorth et al., 2013)	57
3.12. Overview of the web instruction translation	62
3.13. Cucumber cutting experiment	70
3.14. Cucumber cutting – constraint function trajectories	72
3.15. relevant torque directions for maintaining contacts.	80
3.16. Lever between contact point and zero-torque rotation axis.	82
3.17. pancake flipping features	83
3.18. touch-down phase	83
3.19. move-under phase	84
3.20. lift phase	84
3.21. turn phase	84
4.1. Coordinate Systems	90
4.2. Geometric features of a spatula and their parallel / perpendicular re- lation graph	92
4.3. Geometric feature relations of an abstract coordinate system.	93
4.4. Reduced geometric feature relation graph	93
4.5. Rounded spatula with only two geometric features	93
4.6. Points on a spatula that are relevant for frame positioning	94
4.7. The geometric functions used to construct feature functions.	96
4.8. Comparison of 'perpendicular' and 'aligned' feature functions	99
4.9. Comparing VKCs and Feature Functions	100
4.10. Visualization of a twist as the instantaneous axis of rotation	103
4.11. Visualization of rotational and translational axes of a task function	104
4.12. Robot joint axes, extracted from Jacobian matrix	105
4.13. Dependent alignment constraints	106
4.14. Sample constraint elimination processes	108
4.15. Discontinuity plot for Euler Angles	111
4.16. A non-trivial orientation where RPY-angles lose their meaning.	114
4.17. Constraints at singular positions	115
4.18. Overview of the control architecture	117
4.19. Controller that keeps a task angle inside a predefined range.	124
4.20. Behavior of the joint limit avoidance.	127
4.21. Manipulability- and Force Ellipsoid	129

4.22. Manipulability Ellipsoid, evaluated in a particular direction 130

4.23. Snapshots of the workspace evaluation 132

4.24. Illustration of force- and position ranges with a contact force 139

4.25. Experimental setup of the spatula alignment 145

4.26. Effect of a zero-torque constraint for spatula alignment 146

4.27. computed friction force for different configured robot stiffnesses 147

4.28. Computed friction coefficient for pushing movement 148

A.1. Sample task functions that are different but dependent 165

A.2. Gradients of Sample task functions 166

Tables

3.1. Constraint-based description of pancake flipping	65
3.2. Translation from high-level constraint to elementary constraints	67
3.3. Relative movement energy in the candidate constraint direction	71
4.1. Number of different Virtual Kinematic Chains	91
4.2. The parallel / perpendicular relations between feature directions. . . .	92
4.3. The constraints that are controllable over the center of the oven	115
4.4. Success statistics for the pancake-flipping task	132
4.5. Qualitative translation of position and force ranges into a stiffness. . .	137

Today, more and more robots are working in changing environments where they encounter new objects, tools, clutter and changing tasks at run time. For these robots it is no longer enough to perform the same precise movement over and over, assuming that every object is always placed at exactly the same spot. It is unfeasible to have an expert reprogram them each time an object pose changes slightly. Instead, they must adapt their movements autonomously to changed object poses and even new objects, tools and environments using uncertain and incomplete sensor data while ensuring a successful task execution.



Figure 1.1.: *The robot TUM-Rosie, flipping a pancake*

This adaptation requires an understanding, which aspects of a movement are important and which can be changed without affecting the movement goal. Take the example task of flipping a pancake with a spatula in Figure 1.1, (Beetz et al., 2011): The robot pushes the spatula under the pancake, lifts and turns it so that the pancake falls onto the oven again. During the pushing, it is important (a 'task constraint')

1. Introduction

that the spatula is pushed down firmly onto the oven and the front edge is aligned with the oven surface. During the turning, the spatula must be held above the oven surface. But it is *not* important (a 'task freedom') from which direction the spatula is pushed under the pancake – the pancake is round, after all. Other unimportant details are the exact height of the pancake during turning or the exact rotation axis. These aspects can be altered to accommodate for changed object poses, new obstacles or a different table height.

Humans perform these adaptations with ease as they have implicitly learned them as task constraints and -freedoms. In addition to changes in their environment, humans must deal with significant noise in their motor commands. They use implicit knowledge about task constraints to shape their muscle activation, correcting their movements *where it matters* (Wolpert and Ghahramani, 2000). Because part of the noise remains uncorrected (mainly in the task-freedom direction), human movements vary wildly even in simple laboratory settings (Stulp et al., 2009); yet they are highly successful.

A current mobile robot, challenged with the same task, is much less competent: It can localize the pancake oven on the table and the spatula in it's hand, move it's way-points accordingly and avoid new obstacles on the way. But these way-points are usually rigidly attached to the objects: The robot can't exploit the symmetry of the oven or the pancake. It can't change the lifting height when the oven is on a higher table.

When a task is only specified by it's rigidly attached way points, then the robot is left with a very small work space to adapt it's movements – even for seemingly small changes in the environment, the robot hits it's limits fairly quickly and needs manual reprogramming.

By exploiting the 'task freedom', our robots gain additional flexibility. This flexibility becomes important when the robot is challenged with different objects, tools, environments, or when a plan is transferred to a different robot.

However, it is not clear how task freedom or constraints can be represented for robots in a way that scales to even medium-size tasks.

The goal of this thesis is to represent such task constraints for everyday movements in the robot framework in an intuitive, flexible and declarative way. This representation shall enable a high level planner to assemble movement descriptions, have them executed by the robot's motion controller and reason about their execution. This way, I aim to capture some of the human competence in generalizing it's movements to new situations and define the space in which the robot can smooth and optimize it's movements.

Informally, such task constraints can be described by qualitatively relating objects and tools to each other, referring to their geometric properties like corners, edges, or symmetry axes. A slightly more coarse level of description is commonly found in high level planners, dealing with objects as a whole and 'big-step' actions, without movement details.

This thesis augments such 'coarse' descriptions like "move-under(spatula, pancake)" with a description of task constraints, like "align(spatula-front-edge, oven)" or "push-onto(spatula-blade, oven)". It implements a movement execution environment that can respect these constraints and it provides tools to reason about such constraints. These descriptions serve as a compact inter-lingua between the robots system components to communicate object features, constraints and success or failure of movement (see Figure 1.2)

This deepens the robot's movement understanding on the symbolic side and connects the powerful abilities of constraint based control architectures to the reasoning capabilities of symbolic planners.

This thesis outlines, how this kind of hints can be obtained from partial and abstract information sources like web instructions, robot knowledge bases or simulation experiments. They are also easier to adapt to the situation at hand than other more 'complete' skill descriptions – the simplicity of a single constraint makes it easy to create and test new constraints, one at a time.

This kind of constraints can be executed using control frameworks such as Operational Space Control (Khatib, 1987) or iTaSC (De Schutter et al., 2007), which are based on the Task Function Formalism. These frameworks are capable of performing many challenging tasks in parallel by exploiting all degrees of freedom of a robot. They can also weight or prioritize their tasks. These frameworks, tend to specify more

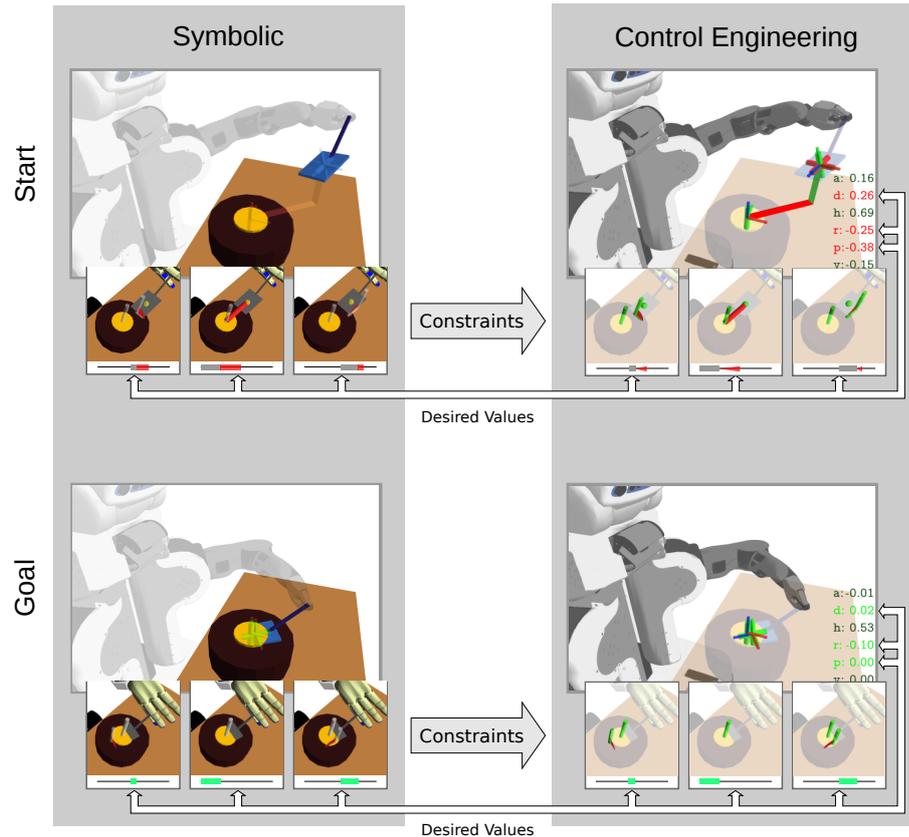


Figure 1.2.: *Bridging of symbolic- and control engineering approaches: Abstract constraint descriptions augment the coarse and qualitative actions on the symbolic side. These constraints are grounded in the robot's perception and concrete geometric computation, so that the controller can execute them. Both sides exchange current- and desired values.*

'complete' task descriptions, in that they explicitly state all constraints and freedoms in a 6D coordinate frame. These descriptions are more coordinate-centric, abstracting away objects and tools.

For specifying task constraints in e.g. the iTaSC framework, several, currently manual steps are necessary: Coordinate systems must be selected, so that the relevant movement directions (symmetry axis, edge direction, etc.) coincide with a coordinate axis. Then, a virtual kinematic chain is spanned from a tool to an object on the scene to a tool in the robot's hand. The translational and rotational degrees of freedom of this chain are then related to the task constraints that were stated in a qualitative way and either a position- or a force controller is chosen for that direction. More complex constraints can be expressed by implementing the forward and inverse kinematics

of a custom 'chain'. The challenges for an automatic selection of coordinate frames and kinematic chains are described in detail in Section 4.1.1. Basically the choice of coordinate frame and virtual kinematic chain are often ambiguous – this kind of modeling calls for a 'complete' picture of the task.

Such an approach is also cumbersome to extend to new objects, tools and tasks: For an object, a programmer must manually annotate every object by placing a coordinate system 'the right way'. For a Skill, he must decide on a kinematic chain and determine constrained and free axes.

The problem is that coordinate systems and kinematic chains are rather large entities, describing not only a single 'simple' constraint but rather a complete task setting, encompassing all six degrees of freedom.

This makes it hard to 1) relate to constraint descriptions like 'keep the spatula horizontal', 2) automatically adapt a movement to new situations and 3) assemble the aforementioned small pieces of information into a consistent movement description.

A robot in a human kitchen has an easier task: It uses two 7-dof arms and two hands to control only two objects. Thus, this thesis can take a simpler, more direct approach to define a controller: It directly uses position and orientation of 'geometric features' on the objects and tools that it handles. Such features include an oven symmetry axis or the spatula front edge, which are localized by the perception.

Then, relations between these geometric features like 'align(spatula-front-edge, oven-surface)' are automatically mapped to one-degree-of-freedom constraints – either a desired position or a desired force along the task direction. This is a much more direct way to generate constraints and it naturally keeps a reference to the used objects, their geometric features and the geometric relations between them. It eliminates the 'artificial' requirement for a coordinate frame and uses the Task Function Formalism to combine these constraints.

The robot is using this frameworks' ability to exploit the remaining degrees of freedom, i.e. to generate feasible robot postures and movements that fulfill all given constraints.

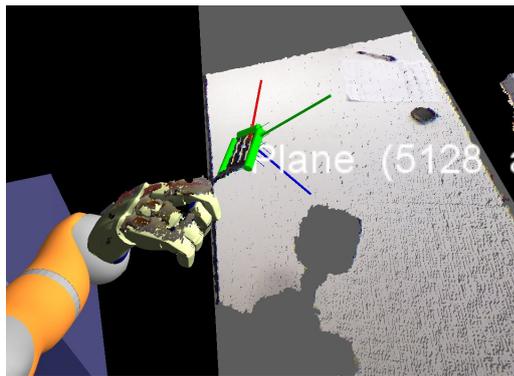
1. Introduction

By introducing this fine-grained movement specification, this thesis lifts the construction of movements to a higher, object-(part)-centric level:

It specifies qualitative relations between objects, grounded in the robot's perception that is suitable for high level planning but encodes a lot of freedom that the robot can exploit and is executable without further intervention. Yet, this description is very detailed and flexible: Both position- and force-constraints can be composed at degree-of-freedom level; this enables re-use and a level of detail only found in controller architectures. It also enables an in-depth understanding of execution performance, opens the door for systematic experiments, and the incorporation of partial information into a movement specification. Further, it relates these fine-granular task descriptions to existing task-function based skills on a degree-of-freedom level – with this ability a robot can take advantage of existing pre-defined skills as well as of partial and abstract hints about movements.

With all these advantages, the proposed motion representation also poses two challenges: The first one is that the perception must localize geometric features like edges, corners or symmetry axes, rather than just 'an object'.

This extra effort makes the localization of objects more explicit: A 'classic' localization of a spatula may place its origin at the center of its blade, aligning the z-axis with the blade normal and the y-axis with the front edge. The coordinate frame placement is done by a convention, which must be reconstructed to perform geometric task planning. In addition, the size of the blade must be remembered.



(a) Line and plane features detected on a spatula

Figure 1.3.: *Detected line and plane features on a spatula*

In the proposed approach, the spatula is analyzed geometrically and front edge, side edges and blade are represented and communicated explicitly, edge sizes are encoded in their geometric representation. Instead of relying on conventions for coordinate frame placement, the object properties are represented directly, can be analyzed and reasoned about. This thesis presents approaches to localize such features in unknown objects and tools.

Another challenge is that the proposed movement representation is so flexible that abstractly and automatically constructed constraints at this level of detail may contradict each other, leading to unpredictable behaviour. For instance, the constraints 'keep the spatula front edge *aligned* with the oven surface' and 'keep the spatula front edge *perpendicular* with the oven surface' could have been specified using contradictory knowledge sources. To approach this problem, this thesis presents tools to quickly and automatically detect such conflicts. It enables a high level movement planner to compose consistent constraint sets.

The contributions of this thesis are:

A flexible constraint-based movement specification for everyday movements that is shared among high level planning, control and perception. It is general, compact, flexible, mostly qualitative and is composed of very simple constraints, each defining a single degree-of-freedom. It provides a simple and effective way to specify both position- and force constraints. It captures only the important aspects of a movement, i.e. those that are necessary for a successful execution, leaving as much freedom up to the robot's controller as possible. Yet, it is general enough to describe any movement that can be represented as a sequence of poses.

The integration of this movement specification with perception, high-level planning and execution. This thesis details, how the geometric features of objects and tools are localized, how a high-level planner can generate and tune constraint-based movement descriptions from various sources, how they are translated automatically into an executable controller and monitored by the planner. This shows the advantages of using a detailed, shared movement description throughout the robot control architecture rather than tucking away the details into the movement controller component.

Reasoning tools to support the composition of constraints, to ensure a consistent movement description and to compare these movement descriptions with existing task-function

1. Introduction

based skills. These reasoning methods can detect conflicts between two sets of constraints on various levels. They are implemented in a library that can be used by the planner and are based on the same functions that are used by the movement execution controller. The same method is shown to enable comparison with existing skills on a degree-of-freedom basis: If a constraint in the new description matches a degree of freedom of another task-function based skill (which can be implemented in an arbitrary way) then our method will find this match.

The movement specification has been validated on several challenging manipulation tasks, both in simulation and in the real world, using two high-class, compliant manipulating platforms. It has proven its robustness in demonstrations under public review.

The advantages of constraint-based movement control for every-day manipulation are verified experimentally, showing that they significantly increase the work space for these actions.

In Section 4.1.4, the constraint reasoning methods are illustrated for several practical examples, showing how a conflict-free constraint specification can be constructed and verified. The proposed force specifications are evaluated with two experiments, demonstrating the effect of a zero-torque constraint for aligning a spatula with an oven and the effect of friction during manipulation.

Conceptual Framework

This chapter gives an overview over the movement specification language, explains the design decisions and compares the parts of this language to related approaches. It describes how the framework tightly couples high level reasoning, perception and the control architecture and outlines, how such a specification can be generated by integrating various information sources available to high level reasoning, perception and monitoring.

In order to describe a movement, I define *geometric features* like 'spatula front edge', 'blade normal' or 'center of the pancake', create geometric relations – *feature functions* – like 'distance' or 'alignment' between these geometric features and set constraints on the *values* of these feature functions which are solved using an appropriate controller.

Consider the exemplary constraints in Figure 2.1: The robot must turn the yellow pancake on the brown oven using the grey spatula. This movement is split into three phases:

1. Pushing the spatula under the pancake
2. Lifting the pancake
3. Turning the spatula until the pancake falls back onto the oven

During the push phase the robot needs a constraint that restricts the distance of the spatula from the oven center and places the spatula on the oven surface and *next to* the pancake (but not on top of it, *distance* constraint). Before the pushing, the front edge of the spatula must be well aligned with the oven surface, so the pancake does

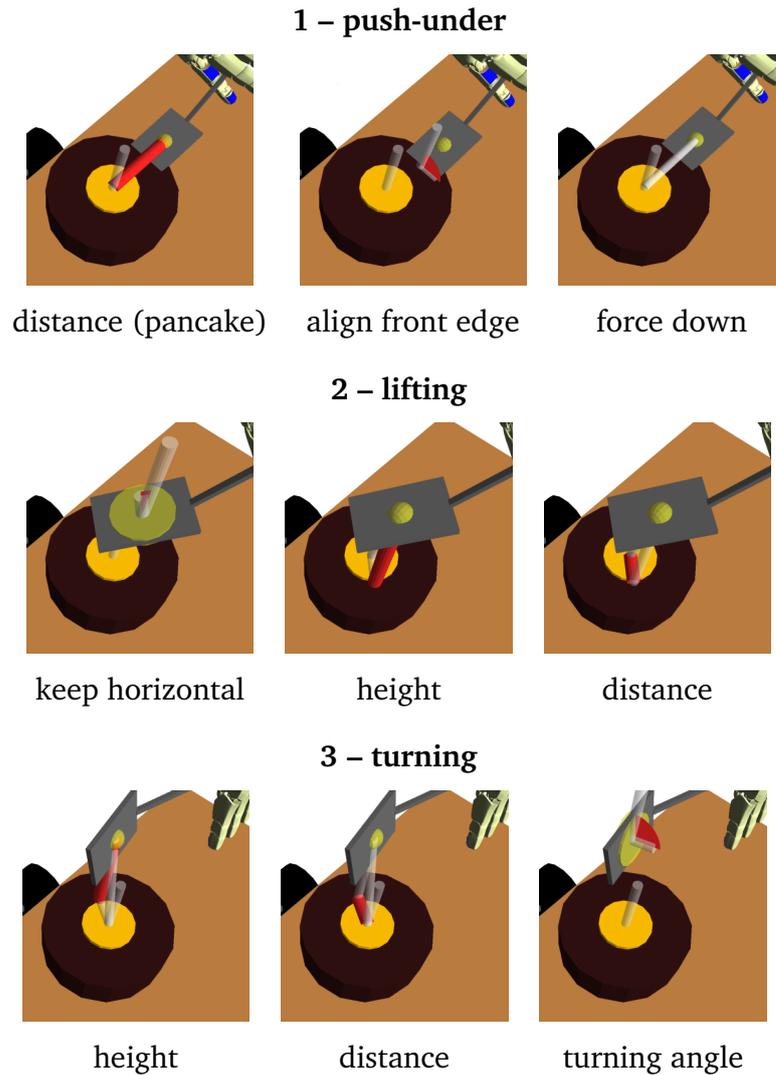


Figure 2.1.: *Pancake flipping task specified as constraints*

not get destroyed (*align* constraint). The robot should exert a downward force on the spatula in order to maintain contact during the pushing (*force down* constraint). When all these constraints are fulfilled, the robot can move the spatula to the center by reducing the distance between spatula and oven center.

Once the spatula is well under the pancake, the robot can lift the spatula (*height* constraint), while maintaining a more or less horizontal orientation of the spatula (*horizontal* constraint) and keeping the spatula over the oven (*distance* constraint).

When spatula and pancake are in a comfortable position over the oven, the robot can begin to turn the pancake: It turns the spatula such that the blade faces down and gravity can pull the pancake back onto the oven (*turning* constraint). During this phase, the robot must keep the spatula over the oven (*distance* constraint) and high enough so that the spatula does not crash into the oven (*height* constraint). In fact, *horizontal* and *turning* are controlled by the same task function, which is set to different values.

2.1 Definition of the Movement Specification Language

In the following, I put these informal statements into more precise terms, introducing the constraint-based movement description that is used throughout this thesis. In addition, the rationale of this description is explained.



Figure 2.2.: *Movement phases for the pancake flipping task*

A movement is modelled as a partially ordered sequence of movement phases (\mathcal{P}, \preceq) , which induces a directed acyclic graph as shown in Figure 2.2. Each phase $p \in \mathcal{P}$ consists of two sets of constraints $(\mathbb{C}_{dur}, \mathbb{C}_{end})$: one set \mathbb{C}_{dur} that shall be fulfilled during the movement phase and one set \mathbb{C}_{end} that shall be fulfilled at the end. A movement phase ends when all the constraints $\mathbb{C}_p = \{\mathbb{C}_{dur}, \mathbb{C}_{end}\}$ of a phase p are fulfilled. In the pancake-flipping example, the following constraints form the first movement phase 'push the spatula under the pancake':

- $c_{dist-pancake}$: 'keep the distance to the center of the pancake smaller than the radius of the pancake'
- $c_{dist-oven}$: 'keep the distance to the center of the oven smaller than the radius of the oven'
- $c_{align-front}$: 'align front edge with oven'
- c_{force} : 'push down onto oven'
- $c_{horizontal}$: 'keep the spatula horizontal'

So the first two movement phases are

$$p_1 = (\mathbb{C}_{dur} = \{c_{align-front}, c_{force}\}, \mathbb{C}_{end} = \{c_{dist-pancake}\})$$

$$p_2 = (\mathbb{C}_{dur} = \{c_{dist-oven}, c_{horizontal}\}, \mathbb{C}_{end} = \{c_{height}\})$$

When all constraints of a movement phase are fulfilled, then the next phase begins. This condition serves as a time-independent trigger for the sequential control and conveniently encodes goal distances into the constraints.

A single constraint is a 5-tuple $(f_A, f_B, f_f, p_{des}, f_{des})$ that consists of two 3D geometric features f_A and f_B , a *feature function* f_f and desired ranges for the position p_{des} and force f_{des} .

The geometric features are, for example, edges or corners of objects and tools that the robot handles and are represented by a position $pos \in \mathbb{R}^3$ and a direction vector $dir \in \mathbb{R}^3$, i.e. $f_{\{A,B\}} = (pos, dir)$. The feature function f_f uses their relative pose to compute a scalar value, like a distance or angle: $f_f : SE3 \times SE3 \rightarrow \mathbb{R}$. The desired position- and force ranges p_{des} and f_{des} specify a desired value range for this scalar value and a desired force that shall be measured along the direction that is defined by the feature function.

The following subsections describe the geometric features, feature functions and desired ranges in detail, explain the rationale and design choices and point out the benefit for the robot framework and for the components perception, planner and knowledge base.

2.1.1 Geometric Features

The *geometric features* precisely describe 'what' must be manipulated. They are symbols which describe the movement-related object- and tool parts like a blade or front edge but also an object's symmetry axis.

I choose simple point- plane- and line features as depicted in Figure 2.3, consisting of a position $pos \in \mathbb{R}^3$, a direction vector $dir \in \mathbb{R}^3$ and a size which is encoded into the length of the direction vector: $size(f) = ||f.dir|| \in \mathbb{R}$. This can be e.g. the length of a edge, the size of a surface patch or the length of a cylinder. These features are detected automatically by the robot's perception system, like in Figure 2.4 (see Section 3.1 for details) and are suitable to describe the motion-related aspects of most objects with relatively few features.

2. Conceptual Framework

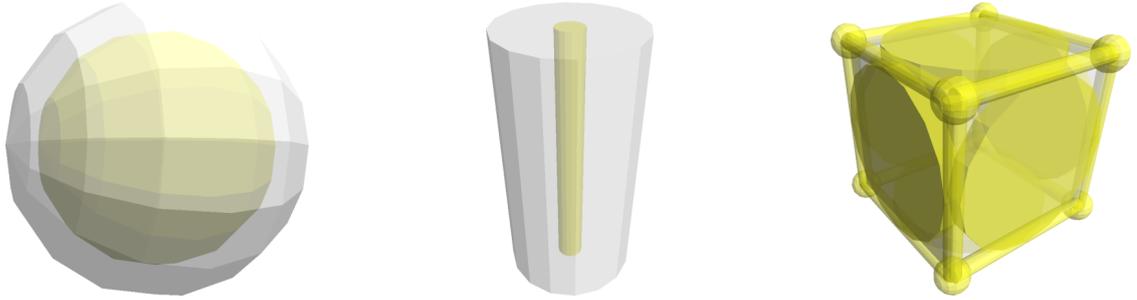


Figure 2.3.: *Geometric features that are used for control: A sphere is represented as a point with radius (left), a cylinder is represented as a line segment plus radius (middle) and a cuboid is represented by its corners, edges and faces (right).*



Figure 2.4.: *Geometric line features that were detected in a long spatula and a skimmer.*

The coordinate system for the mentioned position and direction vectors is located in the robot's tree of known coordinate systems like 'right hand', 'camera' etc. An infrastructure is assumed that can transform between these coordinate systems.

On the one hand, these geometric features provide a fine-enough granularity of object description so that movement constraints can adequately describe *how* an object is used, e.g. “the spatula front blade must be aligned with the oven surface”. They facilitate modular constraints that each describe a single aspect of a movement. On the other hand, these features are compact and easy to handle for the movement controller, the symbolic planner, and a Prolog-based robot knowledge base (Tenorth and Beetz, 2009).

They are also a good match for geometric reasoning: Properties like near, parallel, perpendicular, adjoined, etc. are easily expressed using their positions and directions. Such statements are part of a semantic object description as described in (Tenorth et al., 2013).

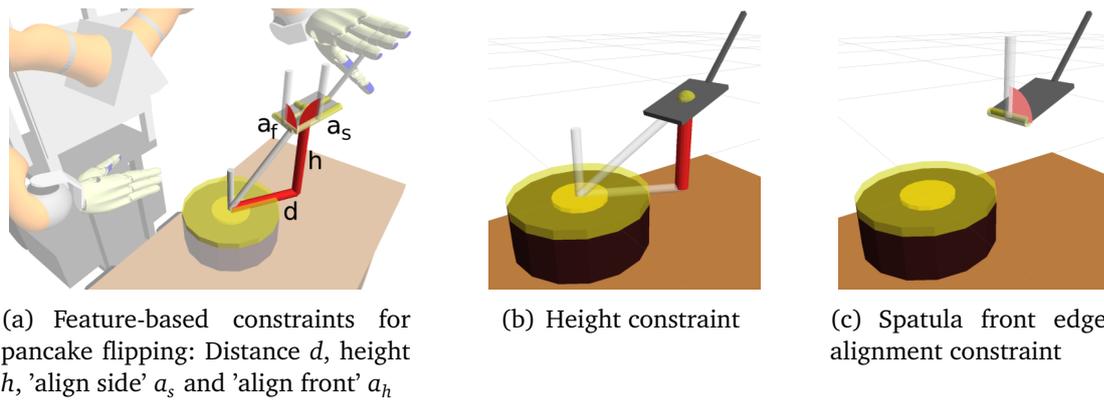


Figure 2.5.: *Feature Functions for the pancake flipping task*

Compared to more 'complete' models like point clouds or meshes, they do not require storing large amounts of data, which suggests a 'scene memory' with the associated housekeeping chores like deleting old object instances that are no longer needed. Such representations may be more appropriate for some object-level constraints like 'distance between two objects'.

2.1.2 Feature Functions

The next 'ingredient' for the movement representation is the *feature function* f_f , which describes the spatial relation between two geometric features f_A and f_B . Examples are the distance between spatula blade and pancake or the angle between the spatula front edge and the oven surface. They are the symbolic vocabulary for spatially analyzing a scene and describing what must be *where*.

A feature function is a one-dimensional task function over two geometric features f_A and f_B or, more specifically, their relative pose in space $\mathbf{x} = \mathbf{x}_{f_A}^{-1} \mathbf{x}_{f_B}$, where $\mathbf{x}_{f_A}, \mathbf{x}_{f_B}, \mathbf{x} \in SE3$:

$$f_f(\mathbf{x}) : SE3 \rightarrow \mathbb{R}$$

As these functions are used as *task functions* in the sense of (Mason, 1981), they must be differentiable, i.e. their derivative must be well defined.

The feature functions used in this thesis can be understood as compositions of projections, vector length and -angle functions. The input to these functions are the position

2. Conceptual Framework

and direction of the involved features and the final scalar value is either a length or an angle. Figure 2.5 (a) shows the feature functions used for pancake flipping: Figure 2.5 (b) visualizes the height of the spatula above the oven as the vector between oven and spatula, projected onto the normal vector of the oven surface. Figure 2.5 (c) shows the alignment function of the front edge of the spatula with the oven surface. However, in this thesis, for the purpose of both planning and control, the feature functions are considered as opaque.

A feature function that is bound to two concrete geometric features, i.e. a tuple (f_f, f_A, f_B) , defines a movement direction for the robot: For any pose of one geometric feature relative to the other, the feature function's value states, how well a symbolic constraint is fulfilled. The gradient of this 'error' function defines the direction where this value becomes bigger or smaller. The null space, i.e. the Cartesian directions that are not mapped into this task space, are free for the robot to optimize. At this level, conflicts between different (sets of) constraints can be determined. This can be used to automatically select a minimal set of feature functions or decide whether two different movement descriptions tackle the same aspects of a movement. This is discussed in detail in Section 4.1.4.

Also for the choice of feature functions I strive for simplicity, in order to avoid over-specification and to enable reasoning. Take the example of the constraints 'keep horizontal' and 'turn': These can be represented by the angle between blade and gravity (one constraint) or the horizontal alignment of front- and side edge (two constraints). While the two constraints allow to more closely control the robot (which was needed during the push-under phase of the pancake flipping), it also restricts robot movement more than necessary during the turning phase. The simpler angle-with-gravity constraint allows much more efficient robot movements.

These feature functions are evaluated and (numerically) differentiated in the control framework in real time. In addition, some reasoning tasks evaluate these functions in order to detect conflicts between them. Therefore, the feature functions are implemented as a library that can be used both by the controller and the planner and reference them by their function names. This ensures consistency and gives the planner access to the same movement model that the controller is using. Further details on feature functions can be found in Section 4.1.2.

2.1.3 Desired Ranges

The third important piece of information is the desired values for the feature functions, like desired distance, angle or a contact force. As the flipping example already points out, the robot needs both equality constraints, e.g. for the alignment of the spatula front edge, and inequality constraints e.g. for the minimum height of the spatula during turning or the distance of the touch-down position of the spatula from the oven center.

The movement representation generalizes these equality and inequality constraints into *desired ranges*: When the lower and upper bound of the range is the same then it is an equality constraint, when one bound is infinity, then it is an inequality constraint.

Formally, a desired position range p_{des} is a subset of the *extended real numbers* $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$, specified by lower and upper bounds p_{lo} and p_{hi} :

$$p_{des} = \{p \mid p_{lo} \leq p \leq p_{hi}\}, \quad p_{lo}, p_{hi} \in \overline{\mathbb{R}}$$

In the same manner, a range of desired forces f_{des} is specified along each feature function's direction. By setting these ranges appropriately, either force- or position control is specified while the other variable is a 'safety net'. This formulation abstracts away the usual decision for force- or position control, lifting the description to a more task-related level.

It is a more powerful specification compared to most of the constraint based control approaches presented later in this section, which specify only a single set-point for either force or position.

Symbolically, this part of the description can be encoded in the verb or spatial relation like 'align' 'under' or 'next-to'. When features and feature functions are selected well, these values are often zero, one, or can be derived from feature sizes. Other values like lifting height may be derived from observation. Through simple interval arithmetic, several constraints on the same feature function can be integrated.

2. Conceptual Framework

During movement execution, these ranges must be consistent with the robot's abilities: Perception inaccuracies and robot arm precision must be taken into account, which translate into a minimum size for the position range.

A combination of two geometric features f_A and f_B , a feature function f_f and the desired ranges for position and force, p_{des} and f_{des} , forms an (elementary) constraint. Several of these elementary constraints make up a movement phase, which ends when all of its constraints are fulfilled.

By construction, these constraints naturally specify both, a movement direction and a 'goal distance', that defines when the movement is finished. One or several of these feature function constraints map to verbal spatial relations like 'above' or 'horizontal' or verbs like 'turn' or 'push down'. By the choice of words, these relations usually contain the information whether a constraint c shall be fulfilled throughout a movement phase ("while pushing down...", "keep horizontal"), $c \in C_{dur}$, or whether it is the goal of the movement ("move", "turn", "push"), $c \in C_{end}$. In the former case, an inappropriate movement shall be avoided while in the latter case a movement is demanded. The feature function based constraints can represent both, and the execution side is exactly the same – what differs is merely the expectation for the monitoring.

The presented movement representation leaves open how exactly the movement shall be performed; the only assumption is that the movement execution is 'reasonably smooth' i.e. no strong accelerations occur. The exact trajectory is subject to optimization, which is further explained in Section 2.4.2.

2.1.4 Execution

The robot executes such a movement using a simple controller/trajectory generator in feature function coordinates that tries to move the robot inside both the desired position and force ranges, using a roughly trapezoidal velocity profile. The generated task function velocities are then transformed using a 'Feature Function interaction matrix' \mathbf{H} which is computed numerically and represents, how much the values of the feature functions change, given a Cartesian movement of the features:

$$\mathbf{H} = \frac{\partial f_f}{\partial \mathbf{x}} \quad (2.1)$$

i.e. this matrix transforms Cartesian velocities $\dot{\mathbf{x}}$ of the object or tool into the task-related feature function velocities $\dot{\mathbf{y}}$:

$$\dot{\mathbf{y}} = \mathbf{H}\dot{\mathbf{x}}$$

The robot Jacobian \mathbf{J}_R transforms robot joint velocities $\dot{\mathbf{q}}$ into these Cartesian velocities:

$$\dot{\mathbf{x}} = \mathbf{J}_R\dot{\mathbf{q}}$$

so combined, they relate robot joint velocities to feature function velocities:

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{J}_R\dot{\mathbf{q}} \tag{2.2}$$

Solving this equation for $\dot{\mathbf{q}}$ yields:

$$\dot{\mathbf{q}} = (\mathbf{H}\mathbf{J}_R)^\# \dot{\mathbf{y}} \tag{2.3}$$

where $(\mathbf{J}_R\mathbf{H})^\#$ is a damped least squares pseudo inverse.

The interaction matrix \mathbf{H} is also used by the planner to determine whether two constraints are independent: If \mathbf{H} is rank deficient, i.e. its rank is lower than the number of constraints, then not all constraints can be controlled independently.

2.1.5 Integration into Robot Software Architecture

As an example, Figure 2.6 shows the description for a constraint that keeps a spatula *above* an oven, using a *distance* and a *height* feature function.

In this thesis, I often simplify this notation to the form in Figure 2.7: I replace the coordinate description of the geometric features with an image and write the constraint as a function of these features. The desired ranges may be expressed as equalities and inequalities.

This movement description integrates the robot software components (perception, symbolic planner and movement execution) much more tightly than previous approaches: The *perception* system localizes not only objects as a whole but it decom-

2. Conceptual Framework

```

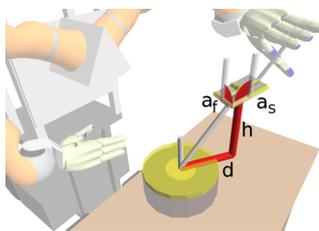
features{
  feature{
    name = 'spatula-front-edge'
    ref_frame = '/spatula_blade'
    type = LINE
    position = [0.0, 0.0, blade_length/2.0]
    direction = [0.0, blade_width/2.0, 0.0]}
  ...}
constraints{
  constraint{
    tool_feature = 'spatula-front-edge'
    object_feature = 'oven-center-point'
    function = 'distance'
    position_range [0.0; oven_radius]
    force_range [-10N; 10N]}
  constraint{
    tool_feature = 'spatula-front-edge'
    object_feature = 'oven-plane'
    function = 'height'
    position_range [0.1; +inf]
    force_range [0N; 30N]}
  ...}

```

Figure 2.6.: Constraints for 'above' relation

poses them using geometric features, either using its optical- and 3D-camera or a CAD model analysis.

The *symbolic planner* assembles the movements at a relatively detailed level: It not only determines the overall structure of the movement, i.e. the movement phases and the constraints that make up a phase. It also considers the geometric features, relates them to each other using feature functions and determines desired ranges. It is the component that has the most convenient access to the robot's knowledge and the one that can trigger information-gathering processes. Therefore, this component was chosen to combine all the small pieces of information that were gathered from experience, common sense knowledge and various web resources. This thesis will show how quite a few pieces of information from these resources match well with the details of the proposed movement description.



- (d) $\text{distance}(\text{spatula}_{\text{front-edge}}, \text{oven}_{\text{center}}) \in [0, \text{oven}_{\text{radius}}]$
- (h) $\text{height}(\text{spatula}_{\text{front-edge}}, \text{oven}_{\text{plane}}) \in [0.1, +\infty)$
- (a_f) $\text{align}(\text{spatula}_{\text{front-edge}}, \text{oven}_{\text{plane}})$
- (a_s) $\text{align}(\text{spatula}_{\text{side-edge}}, \text{oven}_{\text{plane}})$
- (p) $\text{pointing_at}(\text{spatula}_{\text{axis}}, \text{oven}_{\text{center}})$

Figure 2.7.: Constraints for 'above' relation, simplified

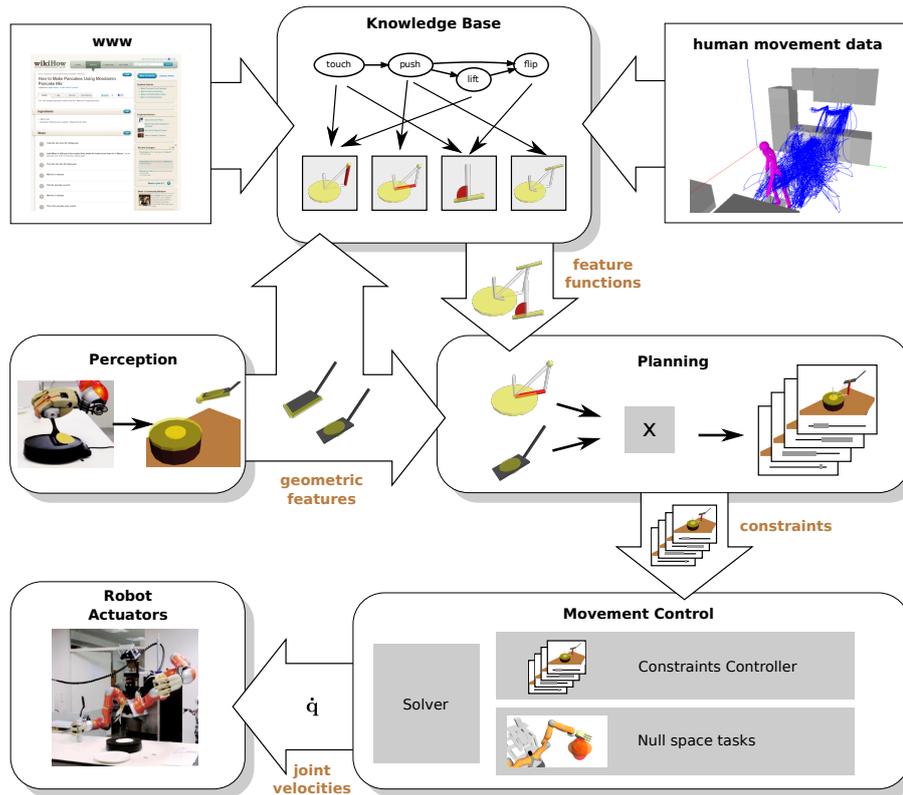


Figure 2.8.: *Overview of the system architecture: The Robot's knowledge base contains abstract constraint specifications using feature functions for everyday tasks, partially extracted from web- and human movement data. The perception localizes geometric features in objects and tools. The planner combines geometric features and feature functions into concrete constraints and sends them to the controller for execution.*

Using this detailed constraint description, the *movement execution* knows precisely, which movements are allowed and can react competently to disturbances. It can also give detailed feedback on the current state of execution, which the planner will understand easily.

2.2 Related Concepts

Having introduced the movement description, this section discusses related approaches and points out the differences to the presented movement specification.

2.2.1 Related Movement Representations

A number of paradigms have been developed for specifying the activities to be performed by autonomous robots. These paradigms include behavior-based control (Matarić, 2001), emergent and developmental approaches (Metta et al., 1999), control-theoretic (Khatib, 1987; Smits et al., 2009) and symbolic methods (Cambon et al., 2004).

With symbolic approaches, movements are programmed in terms of objects, desired effects and undesired side effects. They can effectively represent large-scale tasks, including recovery from failures. One example that introduces some structure in the movements is (Müller, 2008). It decomposes movements and computes intermediate poses like approach, retract, carrying-pose, etc. Each pose is computed by its own, hand-coded, object-dependent heuristic. At this level, all object descriptions, and the complete scene- and task context is available for every single movement phase. But the task constraints for each phase, i.e. how an intermediate pose may be modified, is implicitly encoded into the mentioned heuristics. Failures are reported in the form of 'out-of-reach' or 'collision' events.

This implementation is difficult to extend in a general way: It takes a programmer who must read the existing heuristic functions, understand the geometric computation and adapt them to a new object. Automatically modifying a heuristics is difficult, too: Even if it had suitable tuning parameters, the feedback on failures does not tell whether a change in the movement was a step towards or away from success.

My approach does allow such incremental improvement: It offers a uniform way to modify a movement by adding or removing constraints or relaxing / tightening them. When a constraint is violated, it always reports by how much.

Control engineering approaches, on the other hand, can deal well with disturbances, ensuring that positions and exerted forces are maintained according to the task spec-

ification. They encode the task constraints explicitly: By placing coordinate systems and restricting the transformation between them, they analytically set up a task-related coordinate system that in my approach is spanned by the geometric features and feature functions. The desired value for every task direction is often a set-point for every coordinate direction that is either controlled by a force- or torque controller. This is a special case of my proposed position and force ranges.

These approaches have a principled way to represent task constraints which is flexible and intuitive to understand. However, at this level, the task- and scene context are mostly lost and the object descriptions are reduced to coordinate systems. These coordinate systems and controller set-points are usually set up manually. This abstraction makes these approaches cumbersome to generalize and transfer to new tasks.

Instead of requiring a manual task specification, learning-based approaches extract the task-relevant aspects of a movement from demonstrations (supervised learning, (Calinon et al., 2007; Khansari-Zadeh and Billard, 2011; Khansari-Zadeh, 2012)) or from the robot's own attempts and an end-state goal description (reinforcement-learning, (Kormushev et al., 2010)).

Some of these approaches extract a Gaussian model on the position and dynamics of the movement which encodes, how much variance was seen during the demonstrations. They often perform a dimensionality reduction which – compared to geometric features and feature functions – is fully automated but less transparent. They exploit their learned movement variance to deal with disturbances during execution.

These approaches require much less expert knowledge to program a new task. Unfortunately, it is a challenge to separate the constraints that are related to the task from the constraints related to the teaching environment. For example, considering the task, the robot may be free to move in a certain direction, but the taught movements do not exhibit variation in this direction due to an unrelated obstacle in the area. In addition, the learned result is often both hard to influence and hard to inspect.

It is therefore still difficult to put the learning results in the right context and (partially) transfer them to new situations. Currently, the scope for most learning methods is limited and does not scale well to human-scale tasks.

2. Conceptual Framework

In section 3.2.4, this thesis explores a way to exploit movement data to select feature functions and to determine the desired ranges. This procedure allows us to influence the learning process through a selection of features and task functions and through relaxation and removal of learned constraints.

It has been tried before to integrate symbolic and control paradigms. In so-called 3T architectures, three layers of abstraction were introduced: The top layer contained the abstract, symbolic action specification and the bottom layer contained the control engineering method. As these paradigms were hard to interface, an intermediate layer was introduced that had to bridge the differences of the two paradigms. This layer triggered the start of movements, managed their parallel execution and handled their failures. It inherited the problems of both layers: symbols without plan- and scene context from the symbolic side and abstract coordinate systems and functions from the control side, without an understanding of their inner working.

The intermediate layer constituted a shallow integration, designed to leave the symbolic- and control layer unchanged. This strategy turned out to be of limited use: The symbolic layer was lacking a notion of the concurrent task execution or the scene context. It had no means to understand the behaviour that was generated by the control layer, diagnose failures or to generate suggestions for improvements – the reason for success or failure of a motion was bound to be 'lost in translation'.

A more integrated approach is detailed in (Mösenlechner and Beetz, 2013), where hierarchical action plans use *designators* to specify objects, locations to stand, put down objects etc. These variables are specified using first order logic with predicates like 'left-of' and 'near', but also 'reachable', 'visible' or 'stable'. Some of these constraints use an integrated simulation based on OpenGL and Bullet. Other constraints define a probability distribution over possible locations. In order to resolve a designator at execution time, samples are drawn from the 'sampling'-constraints and are checked against the constraints that are slower to compute. This approach encodes a lot of the freedom that the task has, in a human-readable, symbolic form. It's only shortcoming is that these constraints can't (currently) be used to react to disturbances at execution time. For the sampling-based constraints, however, it would be quite possible to re-implement their defining potential functions in the control module and to differentiate them numerically. Whereas these designators mainly specify object- and robot goal poses, this thesis complements them by focusing on the movements themselves

that reach these goals. In terms of this predicate logic, each movement phase maps to a conjunction of constraints on the (intermediate) goal pose and a conjunction of constraints on the movement that leads to this goal pose.

The work of (Jäkel et al., 2010) uses Programming by Demonstration (PbD) to obtain movement constraints on a similar level of detail: Several demonstrations are aligned and – depending on the recorded object poses, geometric objects like spheres or cylinders are fit around the trajectories. These objects represent, how much the robot may deviate from straight-line movements without risking movement success. On a higher level, this approach also segments the movements and builds a network of movement phases, represented as a disjunctive temporal constraint satisfaction problem (TDCSP). These movements are then executed using a probabilistic planner. This work shows that – given good-enough sensor data – constraint-based manipulation strategies can be learned from demonstrations. It's RRT-based planner is certainly more suitable for situations with heavy clutter. However, online reaction to disturbances require a control-based approach.

This thesis directly introduces a detailed movement representation to the symbolic planner, that captures the control layer's details, specifying how to perform a movement in terms of object parts, geometric relations between and desired position- and force-values. It endows the planner with the means to create, monitor and understand movements in a way as detailed as the control layer does.

In (Somani et al., 2015) the author specifies robotic pick, place and assembly tasks using constraints on CAD models. Like in this thesis, he uses the distance- parallel and coincidence constraints on points, lines and planes and translates these specifications into robot programs. While such constraints are not new to CAD software, this work further proves the applicability of this kind of constraints for robot control. This application did not use force- or inequality constraints.

2.2.2 Geometric Features

This thesis uses 3D geometric features such as points, line and planes on tools and objects, which are localized using a combination of 3d camera and a high-resolution camera. In addition to this method, which is presented in Section 3.1, a number

2. Conceptual Framework

of other approaches might provide useful input for manipulation-related object features:

Detection of object parts in the form of lines and superquadrics has been used before in object classification (Stark et al., 1993) and tracking applications (Sminchisescu et al., 2005). The former work focused on the semantic decomposition of objects in a camera picture using superquadrics, like the decomposition of a hammer into handle and head. The goal of the tracking application was to gain robustness by adding new features to the object model as they become visible. Neither of these approaches has been tested for robotic manipulation so far.

Other approaches use learning to find grasp points or tool tips by their visual appearance (Saxena et al., 2008; Kemp and Edsinger, 2006). These approaches have been very successful for novel objects and have been tested extensively on real robots. These simpler (point-) features seem to be useful in my framework as well, although points alone would not be expressive enough to specify all the movements presented in this thesis.

Visual Servoing as in (Marchand et al., 2005) uses lines and corners in 2D image space to express a desired object pose. The level of detail is very similar to this thesis. However with Visual Servoing, the robot's camera pose is closely linked to the movement description, as the perceived lengths and angles depend on the robot viewing angle: An alignment constraint, for example, is not even visible from certain directions. While the camera positioning may be an important aspect for successful execution, it depends on the robot embodiment, which I seek to factor out of the movement description per se.

In conclusion, from these alternatives, only the simple 3D geometric features fulfill all my requirements for the movement description, but other alternatives may complement the model of the objects and tools that the robot handles.

2.2.3 Feature Functions

A *feature function* describes the spatial relation between two geometric features, like distance or alignment. It is the symbolic vocabulary for spatially analyzing a scene

and describing what must be *where*. A set of these feature functions together with geometric features defines the directions in which the robot shall move, must not move and where it has freedom to optimize.

This representation of a task space was inspired by the Task Frame Formalism, which was first described by Mason in (Mason, 1981). In this approach, a coordinate system – the Task Frame – is placed so that its six axes are aligned with task-relevant directions. Then a position- or a force-controller is assigned to some or all of the degrees of freedom (hybrid control). This is a more complete task description which forces the programmer to decide about freedom or constraint for every translational and rotational direction.

This concept is used in (Bruyninckx and De Schutter, 1996; Khatib, 1987; Prats, 2009), among many others. A large body of work uses this formalism for hybrid position / force control and sensor integration in general, see e.g. (Hayati, 1986; Raibert and Craig, 1981; Vukobratovic and Tuneski, 1994; Kroger et al., 2004).

A classification of the free/constrained combinations in the Task Frame Formalism was given e.g. in (Morrow and Khosla, 1997), where 12 different primitives are identified – many of them are contact formations.

Considerations on the controller setup when the task frame is attached to the hand, the object or some other measured frame, are given in (Kröger et al., 2004).

The work of (Prats, 2009) applies the Task Frame Formalism to tasks like pulling a handle or operating a crank. It also addresses the grasp selection according to the freedoms of the task at hand: Every grasp and every object or handle has a coordinate frame attached, with the free and constrained directions marked. The concrete grasp and movement is then assembled using a set of heuristics which combine grasps and objects / handles. In this thesis, these heuristics are formulated in symbolic, high level statements and are more fine-grained than the task frame-based descriptions.

The difficulty for the Task Frame Formalism lies in the automated placement of such a task frame, given an arbitrary set of symbolically described constraints. Because all degrees of freedom are included in this description, it is also not straight-forward to answer whether a set of symbolic constraints would 'fit' into a single task frame. An

2. Conceptual Framework

attempt to obtain a suitable description for the related *Virtual Kinematic Chains* is described in detail in Section 4.1.1.

Other constraint-based control frameworks focus on the control side (e.g. (Khatib, 1987; Bruyninckx and De Schutter, 1996; Mansard et al., 2009)), leaving the specification of coordinate frames and free / constrained axes as an easy but manual task.

The Feature Frame Formalism in e.g. (De Schutter et al., 2007) extends the Task Frame Formalism so that the controlled axes no longer have to meet in a single point. Applications of this extension are laser pointing tasks or minimally invasive surgery. However, the automated translation from incomplete symbolic constraints is even more challenging.

An even more flexible method to specify constraints has been developed in (Aertbeliën and De Schutter, 2014): 1D-constraints can be composed using 'expression graphs', using a comprehensive set of operators like difference, norm, angle, dot product, cosine, etc. These expressions are specified in the general-purpose programming language LUA, which makes this approach easily extensible.

This thesis uses a subset of these expressions (angle, length, vector difference and dot products / projections) to explain a small number of feature functions which are mapped onto high-level symbols. These few functions turn out to be powerful enough to express a large class of tasks, but it is still feasible to perform a blind search over all combinations of features and feature functions. Such a search, performed over all possible expression graphs would be prohibitively expensive.

Using Visual Servoing, as done in (Marchand et al., 2005; Hager and Toyama, 1998), fine-grained constraint specifications are done in image space: aligning points and lines with each other. Most often, these approaches are used with equality constraints, although e.g. (Dodds et al., 1999) offers an algebra for Visual Servoing tasks that allows to position a robot in a measurement space set up by visual features. Compared to the direct specification of features and feature functions in 3D, the 2D projection, inherent in Visual Servoing, poses extra challenges for a movement specification: It must consider if and when this projection leads to ambiguities and what further constraints on the robot's viewing angle are necessary.

2.2.4 Constraint Ranges

In contrast to ranges of desired positions and forces, specifying single desired values for position or force has been explored extensively in the literature, e.g. in the task frame formalism as in (Prats, 2009; Bruyninckx and De Schutter, 1996), for tasks like opening doors, pushing buttons or actuating mechanisms.

When contact formation is considered, then a force for maintaining contact should be derived; Although contact formations have often been formulated in the Task Frame Formalism as well (e.g. in (Morrow and Khosla, 1997)), this representation is easily split up into feature constraints.

By specifying ranges rather than single desired values, there is a lot more space for optimization. This has also been exploited in (Ogren et al., 2012), where the authors use linear programming to formulate tasks as a set of scalar inequalities in a dual-arm setting. Compared to the pure inequality formulation, my position/force range-approach is somewhat more complex, but it encodes force- and position constraints into one compact representation.

2.3 Generating Movement Specifications

Having presented the movement description and related work, this section now motivates how different components of the robot control architecture – namely perception, symbolic planning and movement execution – are connected by my movement representation. It also sketches, how the robot can exploit different information sources, which are available to these components:

Geometric features are detected by the robot's perception, and constraints are generated by the symbolic planning and the robot knowledge base which has access to sources like web instructions and human movement data. These constraints are executed by the robot's control architecture, which provides detailed feedback that the planner can understand.

This stands in contrast to an approach where a separate movement execution module is responsible for monitoring and improving movements. The rationale is that such a module may not have all the information available that is useful for this task and – more importantly – is not entitled to gather the required information.

The planner, on the other hand, has access to all the knowledge and situational data related to a movement. It is in this component, where the various pieces of information can be compiled into a movement description.

Here, a movement is grounded in perception, and knowledge from various sources is incorporated, like web instructions, a robot knowledge base, human observation or action-effect relationships. The movement representation can use even tiny pieces of information like 'lift spatula by at least 10cm' and attempt its execution.

2.3.1 Perception of Geometric Features

For an autonomous robot, the geometric features on objects and tools must be grounded in the robot's perception, which localizes known objects and detects prominent geometric features in new objects.

It is important to note, that perception for robotics has different requirements than e.g. 'classical' computer vision: In order to successfully perform large-scale tasks,

perception results must be very reliable. In exchange, the robot can repeatedly look at objects under different lighting conditions, it can grab objects to examine them, etc. In the future, a robot must make use of these possibilities to ensure reliable-enough results.

These features are tagged with the object part name such as spatula front edge, spatula blade, etc. They can be linked to the constraints, movements and tasks in which they are involved. They are part of the vocabulary to the high level planner and define *what* (exactly) to manipulate.

By performing this geometric feature localization, the perception takes part in the reasoning for movement control, it performs an analysis step, which is needed to adapt movements to a new object. It delivers candidates for rotation axes, edges and faces to be aligned, potential tool tips, etc. These geometric features are naturally linked to the movement description.

Section 3.1 explains in detail how a robot can obtain the proposed features from a combination of a high-resolution camera and a 3D camera and how the robot can use the presence of geometric features to classify a concrete, unknown tool.

Related approaches for detecting movement-related features in objects have been described in Subsection 2.2.2.

2.3.2 Selection of Feature Functions

In my system, the high level planner not only performs abstract action selection but it also deals with the details of movements, i.e. their definition, adaptation and the monitoring of their correct execution. It is the component, where active perception can be triggered, information can be gathered from web sources and where success or failure of movements is reported to. Here, it is easiest to make a connection between (quasi-) natural-language statements and fine-grained symbolic movement descriptions. In addition, very small pieces of information can be integrated into a movement: The evaluation and test of these modified movement descriptions – integrated into the running task control – fits well with the high level planning component.

2. Conceptual Framework

Many large information sources are mostly made by humans, and are possibly incomplete or contradictory. There is little hope to synthesize a complete and correct movement description out of any single of these sources. My movement representation language is designed to combine all this information into an executable movement. Naturally, a black box approach for movements can not provide this.

I investigate three kinds of knowledge sources: Firstly, I consider natural-language sources like web instructions, common sense knowledge and human instructions. I aim to ground movements in the 'language' of the robot that it uses to describe the world in terms of objects, actions, processes and so on. This may also provide a basis for a dialogue with a human teacher.

After a short introduction on some related work that generates action sequences which express *what* the robot must do, I outline how constraints about movements can be extracted which describe *how* a movement shall be done. These constraints are derived from action verbs and prepositions. I assume that a natural-language parser provides us with triples consisting of an action or relation and two participating objects and then translate them into one or several executable constraints where each constraint relates two features to each other.

I find that most 'natural' instructions found in the web are relatively 'coarse' and need to be complemented with common sense knowledge in some way. This would lead to very precise instructions, meticulously mentioning every movement aspect and the exact sub-parts of the involved objects. Nonetheless, this representation is symbolic and the translation into natural language is straight-forward (see Section 3.2.3).

Secondly, I investigate the use of human movement data: By closely and accurately tracking objects and tools, I evaluate possibilities to do a fine segmentation and identify the 'constraint' directions where most of the movements occur and possibly those directions where humans aim for 'unusual' precision. I show that the movement energy can lead the *choice of Feature Functions* with somewhat promising results. Even if this analysis had yielded no usable results, the trajectories can always be analyzed with a given constraint function to determine the *value range* that the demonstrations have shown. This piece of data is the most tricky one to obtain from symbolic knowledge sources.

A method for grounding such movement data in a kind of abstract action sequences was presented in (Beetz et al., 2010). This kind of analysis is important to detect the context of a movement. Similarly, the work of (Kjellström et al., 2011) generates action sequences from videos, using the hand trajectory and involved objects and works in very unconstrained settings. Both of these approaches have relaxed requirements on the precision of the data. A possible way to understand learned movements is to take the extracted constraints and (try to) translate them back into high-level statements. This would exploit the relations that were used to relate high-level statements to constraints. The Prolog-based knowledge representation facilitates such a conversion without further effort.

Finally, I outline how to incrementally build movement descriptions and test them in simulation against desired and undesired effects. For example, the pancake must not slide down from the spatula, once it is in the air: This is achieved best by a constraint that holds the spatula horizontal. By construction, every constraint that is discovered in this way, is linked to the desired effect that it causes. This strategy allows to systematically test a large amount of constraints. However, if more than a few constraints are required to achieve a desired effect, then the search space becomes very large.

On the other hand, when a working movement sequence is known, it can be used to initialize a set of constraints: First, a 6D task space is constructed using feature functions on observed geometric features based on the involved objects. This selection might be based on the movement energy that the template movement has in this direction (see also Section 3.2.4). Then, the given movement is transformed into this task space and the constraints can be relaxed, one-by-one, checking each simplification step in simulation.

The constraints from all of these sources can be tested independently and combined into the same movement. Since each constraint is grounded in the (parts of the) involved objects, they adapt equally well to a changed environment. The constraint function names form a low-level vocabulary, that is linked to the robots action knowledge.

The feature constraints are the lowest level in a hierarchy of symbolic movement specification, so low, that they translate directly into executable controllers. But they are still high level enough to carry a semantics that is understood intuitively and that

2. Conceptual Framework

connects to various sources of information. They represent a powerful interface to the robot controller while they remain closely related to the reasoning system.

2.3.3 Constraint Ranges

The previous sections tackled the task function, which serves as a coordinate transformation into a space where axis directions can be assigned useful semantics. This subsection describes the selection of desired values for these task function coordinates.

I use intervals or 'ranges' as a data structure to represent both equality and inequality constraints. By using $+\infty$ and $-\infty$ as valid border values, inequality constraints can be modeled.

Determining these values is the difficult problem of bridging the gap between symbolic and numeric task representations, turning the object- and feature sizes, modifiers like 'carefully' or 'firmly' and error margins into a desired value. It is the last planning step where the planner must deal with numbers – features and feature-functions were selected from a predefined set.

This thesis investigates two approaches to filling in these values: A symbolic approach, using verbal descriptions of constraints, and a data driven approach, using observed trajectories.

In some cases, a verbal description implies a desired value: For example, aligning means that the angle between the involved features shall be zero.

Using suitable features and feature functions, this step *should* be simple and understandable, at least for humans. Values like 0, 1 or the sizes of the involved features should suffice for most constraints.

Let us consider a few examples:

- (1) "keep the spatula above the oven" \rightarrow
- $$\begin{aligned} & \mathbf{height}(\text{spatula}_{\text{center}}, \text{oven}_{\text{plane}}) > 0 \\ & \equiv \text{pos}_{\text{height}} \in [0, +\infty) \\ \wedge & \mathbf{distance_2d}(\text{spatula}_{\text{center}}, \text{oven}_{\text{plane}}) < (\text{oven}_{\text{radius}} - \text{spatula}_{\text{radius}}) \end{aligned}$$

$$\equiv \text{pos}_{\text{dist}} \in (-\infty, (\text{oven}_{\text{radius}} - \text{spatula}_{\text{radius}})]$$

(2) "*align front of spatula with oven*" \rightarrow

$$\mathbf{align}(\text{spatula}_{\text{front}}, \text{oven}) = 0$$

$$\equiv \text{pos}_{\text{align_front}} \in [0, 0]$$

(3) "*push firmly down onto the oven*" \rightarrow

$$\mathbf{force_height}(\text{spatula}_{\text{center}}, \text{oven}_{\text{plane}}) > F_{\text{contact}}$$

$$\equiv \text{force}_{\text{height}} \in [F_{\text{contact}}, \infty)$$

The constraint (1) translates into two inequality constraints, one of them involving the sizes of both objects. The sizes are encoded into the features and can thus be accessed easily in the planner. Constraint (2) is an equality constraint. Constraint (3) encodes a force constraint and depends on the stiffness of the robot to execute correctly. Furthermore, the uncertainty in the perception results must be accounted for during execution, leading to offsets and tolerances in the desired values.

When detailed movement data from a demonstration is available, then these values can directly be extracted by transforming the trajectories into the feature function space as described in Section 3.2.4. This is a reliable way to fill all desired values. Depending on the environment where the movement was observed, it may be possible to further relax the extracted constraints, since additional obstacles may have constrained the observed movement.

If several ranges on the same task function coordinate are specified, they can simply be intersected to form a single range. The uncertainty in the feature position and sizes may be added or subtracted, depending on whether contact shall be maintained like in an alignment task or avoided as during lifting of the pancake.

In a related approach, the authors of (Kaelbling and Lozano-Pérez, 2011) perform this planning step by intersecting ranges, they use heuristics for picking a particular point inside those ranges, and do backtracking if this point violates another constraint that is considered at a later point. After this planning step, one particular desired value was chosen, leaving no more freedom to the movement execution.

2. Conceptual Framework

My representation preserves this freedom and leaves the instantiation up to the movement execution, which can react more competently to disturbances.

2.3.4 Specification of Force Constraints

In order to fulfill force constraints, I take advantage of an impedance controlled robot, like the KUKA lightweight arm or the PR2. In this control scheme, the end-effector position and the force that it applies to its environment, are coupled. Applying force thus translates into setting a position offset that depends on the robot's current stiffness. This approach has the advantage, that the controller naturally avoids destroying objects when a collision occurs, and it also avoids that the arm 'runs away' when the desired force can't be met. Like for positions, I specify desired force ranges for force-based tasks. Combined with the position range, I can express a desired position, if the force does not exceed some limits, or a desired force, if the position does not exceed limits. Further considerations on this issue are made in Section 3.3.

I see two major reasons to specify force constraints: First, to avoid damage during a possible collision, or when the robot intentionally makes contact with the environment.

The contact case can be seen as a grasp to change the environment. A good approach to classify a grasp is by the wrenches that it must transmit to the grasped object (Prats, 2009). Forces can often also be understood as a way to maintain a desired contact.

In frameworks such as (Perdereau and Drouin, 1993), force- and position constraints are combined in a way that some directions are force controlled and others are position controlled. However, force and position for a robot are always tightly coupled: When the robot wants to move its arm, it must exert torque using its motors. On the other hand, when the robot wants to exert a force with its end-effector, then its joints will move (except for the idealized case of a perfectly stiff contact and a perfectly stiff robot). This fact is taken into account with impedance controlled robots such as the KUKA light weight robot (Hirzinger et al., 2002) or (in a limited way) the arms of the PR2 (Willow Garage). In this control law, position and force are explicitly coupled by a stiffness K . On these robots, there is also a trade-off between compliance (i.e.

low stiffness) and position accuracy: With a low stiffness, the detected external force becomes indistinguishable from friction in the gearboxes. Under these constraints, I propose to specify a task by both, a desired position range and a desired force range: For a position task, the force range serves as the safety limits so the robot won't destroy itself or the environment. For a force task, the position range guards against the robot 'running away', when a contact gives way. The specification of force constraints using impedance controlled robot arms and the trade-off between accuracy in position or force are discussed further in section 3.3.

2.3.5 Reasoning Support

So far, I have presented a feature-constraint based movement description, and how it can be constructed from various unreliable and incomplete sources. The description offers a lot of flexibility to incorporate all this information, but it also allows to construct 'wrong' or conflicting specifications that can not be satisfied in practice. Obtaining such a result is easy, especially if the construction is partially automated as is proposed in Section 2.3.2.

In order to manage this flexibility, methods are necessary to analyze possible conflicts between constraints and discontinuities in relevant poses. I investigate, how such reasoning methods can be grounded in the movement execution of the controller.

Specifically, I am interested in whether the constraints in a given set can be controlled independently and whether two sets of constraints are 'identical' in the sense that they control the same movement directions.

This gives the planner, which is supposed to compose sets of feature constraints, a basic check to verify if such a set is valid, i.e. conflict-free and thus executable – blindly constructing some movement, trying to execute and receiving a (timeout-) failure, would be a cumbersome approach.

Compared to the movement execution, the planner has a different 'point of view' and different requirements: The controller is running in a loop at several hundred Hz, making use of local linear approximations. It is executing *one* movement at a time and it assumes it to be valid. The planner, on the other hand, is running at some 10

2. Conceptual Framework

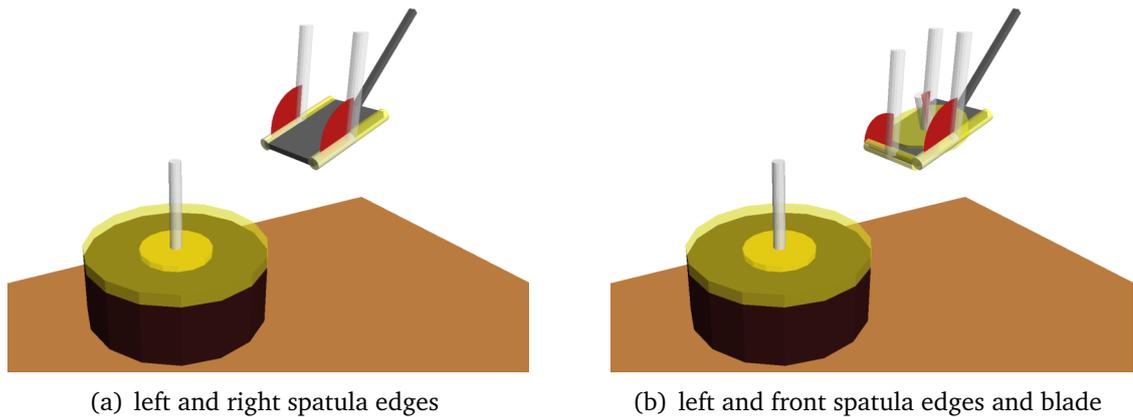
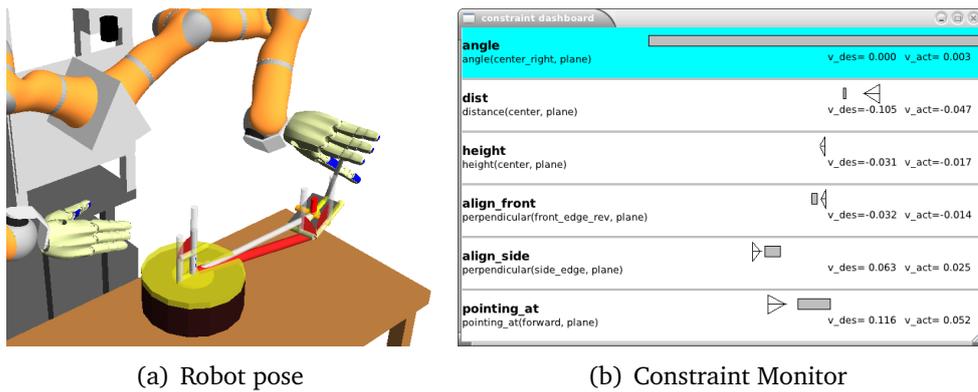


Figure 2.9.: *Dependent alignment constraints*

Hz, making decisions for the next few seconds. It must consider many different constraints, their potential parameters and their effect on the environment. The models for the planner must still be faithful enough so they lead to correct decisions. The manual development of such a model would be tedious and error prone and must be extended when a new feature or feature function is introduced.

Therefore, the feature-based task functions – the core of the movement controllers – are implemented in a library and are accessible to the planning module. These functions are called with a number of random poses and their numerical derivative is computed. By determining the rank of the resulting 'interaction matrix', I determine whether all elementary task functions are independent, i.e. they can be controlled independently. With this method, the planner can check hundreds of movement configurations per second for possible conflicts. But also elementary constraint functions can be unsuitable in certain situations. By numerically computing the second derivative, discontinuities are revealed which are dangerous for the stability of the controller. For example, when the spatula is over the center of the oven, then the direction towards the oven center is undefined and quickly changes with minimal movements. Both of these checks are also implemented in a library and use the same task-function interface as the actual controller. This way, arbitrary task functions, can be checked and compared, not only the proposed feature-based elementary constraint functions. The shared library which contains both, the feature functions and reasoning support, ensures consistency between the planning model and movement execution.

2.3. Generating Movement Specifications



(a) Robot pose

(b) Constraint Monitor

Figure 2.10.: *Monitoring of a pancake flipping task, showing the constraints, their allowed value ranges and current values. The blue constraint is disabled and the arrows indicate the velocity.*

In addition to basic consistency checks, Constraint-based movements also allow for a detailed monitoring. In 'black box' architectures, the movement execution module can only report success or failure. In order to diagnose a failure, the planner needs separate heuristics based on current context and / or extra perception actions. For constraint-based control, a fine-grained monitoring is naturally defined: At any point in time, each elementary constraint reports its current value. By comparing this value against the desired range of values, a Boolean predicate is defined without further effort. This is a detailed and uniform interface to monitor every aspect of constraint-based movement execution. A live monitoring view of a pancake flipping task is shown in Figure 2.10.

2.4 Executing Movements

2.4.1 Control Framework

After defining a constraint-based movement representation and pointing out the means for constructing them, the next subsection describes how to execute these movements.

My constraint description poses a number of requirements for the control architecture that has to execute the movements: A single movement phase is composed of several 1D-constraints that must be fulfilled during this phase, like the distance or height of the spatula in the flipping example. The control framework needs the ability to execute such compositions of constraints, if they are feasible.

Furthermore, the framework must be able to exploit the remaining degrees of freedom of the task in order to adapt to the current environment. The controller must react 'correctly' on disturbances, which might be caused by the unforeseen intervention of a human or the uncertainty in the sensor data. This is especially important when contact is made.

It should integrate with current robot control strategies to be practically useful. Finally, the setup of the controller must be completely automated so that the system can scale to human-size tasks. There are a number of movement control strategies that can fulfill the requirements stated above.

My approach is a control-based architecture based on the Task Function Approach and is shortly explained in Section 2.1: The controller numerically differentiates the feature function f with respect to the feature positions $\mathbf{x}_{f_A} \mathbf{x}_{f_B}$. The resulting *interaction matrix* \mathbf{H} is combined with the robot Jacobian \mathbf{J}_R to compute joint velocities $\dot{\mathbf{q}}$ from feature-space velocities $\dot{\mathbf{y}}$:

$$\dot{\mathbf{q}} = (\mathbf{H}\mathbf{J}_R)^\# \dot{\mathbf{y}} \quad (2.4)$$

The velocities $\dot{\mathbf{y}}$ are computed as a modified P-controller that drives \mathbf{y} into the desired value range.

The interaction matrix captures the directions that are instantaneously important to the task and, conversely, those that are free for optimization. This allows to optimize in the nullspace and pursue secondary goals without interfering with the task.

A lot of research has been dedicated to randomized movement planners like probabilistic roadmaps (Geraerts and Overmars, 2002) or Rapidly-Exploring Random Trees (RRT) (LaValle, 1998). They are a means for finding valid movements in difficult planning situations with many obstacles. Given the right implementation, they can reach probabilistic completeness, in the sense that they eventually find a solution if one exists, no matter how cluttered the environment is. Typically, a random sampling is performed on joint level, the resulting poses are checked for collisions, as are the paths to the neighboring samples. When a connection is found from the starting pose to a pose that fulfills a goal test, then the algorithm terminates with that solution. Then this solution is smoothed and executed on the robot.

Obviously, the feature functions can be used as this goal test, so the robot can exploit the task freedoms in the goal pose. Likewise, inequality constraints can be encoded into the collision tests. This has been done for manipulation tasks e.g. by R. Jäkel (Jäkel et al., 2010). But fulfilling an equality constraint during a movement poses a harder problem as the space of allowed poses becomes a lower-dimensional manifold. Since a random sample will almost surely miss this manifold, extensions are required like the Constrained Bi-directional RRT (Berenson et al., 2009). This extension uses an iterative method to project a random sample onto the constraint manifold. The tasks' degrees of freedom are exploited in the sense that a solution path can move freely in these directions. The undeniable advantage of these methods is the ability to work under heavy clutter. However, the disadvantage is that this method cannot react to a disturbance during execution, which makes them less suitable for movements involving contact in uncertain environments.

The requirement to react 'correctly' on disturbances – that is: such that all constraints are fulfilled – requires an on-line evaluation of the constraints and thus a control framework. Suitable approaches include Khatib's Operational Space Control (Khatib, 1987), iTaSC (De Schutter et al., 2007) and Stack of Tasks (Mansard et al., 2009). All these approaches allow to combine different constraints and execute a task in the null space of another.

2. Conceptual Framework

The task description of Operational Space Control (Khatib, 1987) is similar to the Task Frame Approach, in that it defines positional, force- and free directions in a task-dependent coordinate frame. It is defined on the joint torque level accounts for the dynamics of the task- and null space.

The Stack of Tasks (Mansard et al., 2009) is in use on several humanoid platforms and can execute a deep hierarchy of tasks and insert and remove tasks in this stack during execution. It's task description uses the Task Function Approach defined directly over the joint angles of the robot. This allows to define tasks in Cartesian space as well as in joint space. This framework is currently velocity-based and requires the inversion of the Jacobian matrix.

The iTaSC framework (De Schutter et al., 2007) is another implementation of the task function approach. It also defines a methodology for (manually) constructing a 6D task function that describes a task. In addition, it defines a way of estimating and correcting geometric uncertainties, based on sensor feedback. This is specified in the form of a map relating a state to expected sensor-data, much like the sensor model of a Kalman Filter. Most task functions are described in Cartesian space which provides an interface to separate the task from the robot on which it is implemented. Like the Stack of Tasks, the framework is currently velocity-based.

For my movement execution engine I choose a control framework over a movement planner, since contact with the environment and the control under this condition is an essential part of the envisioned manipulation tasks. The task function interface provides a lean and simple interface, especially the Cartesian variant used in the iTaSC framework, which allows us to separate the task- from the robot kinematics and to easily re-target tasks to other robots, or to two-handed movement execution. Due to my current focus on quasi-static manipulation, I can use the velocity-resolved solver from the iTaSC framework. The robot Jacobian is computed as usual, but the task Jacobian is computed numerically. This simplifies the system and allows to use any differentiable potential function as a task function.

2.4.2 Null Space Exploitation

My movement representation is designed to retain as much of a task's freedom as possible: The features-functions span a minimal task-relevant control space and the desired ranges define all acceptable configurations in this task space. This leaves a big null space which allows the robot to autonomously optimize it's movements.

Such optimization are being done continuously by humans: They optimize for least effort and precision and they can adapt to changed environments without major problems (Mistry et al., 2005). Smoothing and optimizing trajectories in this way can make robot movements more elegant and more understandable for a human observer. However, humans have to deal with much slower signal processing and considerable noise in their actuators (Wolpert and Ghahramani, 2000); it is not clear, whether these optimizations are as beneficial for robots or whether some of them are in conflict with other desirable goals.

In this work, I propose a few more simple null-space tasks which nevertheless make the robot more competent in it's movement execution. Firstly, I make the task possible at all: Due to the extra freedom in the task, the robot can choose from more poses that accommodate it's work space and joint limits. It is therefore successful in more environments. Secondly, the movement can be optimized for smoothness and reduced energy consumption. Instantaneously, this is done by weighting the robot joint movements according to their current joint-space inertia and choose a movement solution that minimizes this effort. However, often this problem is formulated over a complete movement, rather than instantaneously. Another optimization is to prepare for the next movement: In the null space of the current movement, the constraints for the next movement can be executed already. This should lead to more elegant execution. The work in (Keith, 2010) discusses how constraints can be scheduled in a hierarchy of tasks to optimize for execution time. These possibilities are discussed in Section 4.2.3.

2.5 Discussion

The presented approach combines various tools to capture goals and constraints of a movement in a general and abstract way, with the ambitious goal to capture the essence of a motion, i.e. what defines success or failure.

It is very modular and allows to represent even very small pieces of information, collected from various sources. The movement description is easy to interpret, both for humans and the robot controller. And it is independent from the robot's embodiment, as the constraints are grounded in the involved objects.

It is a good description for the forbidden movement directions. For the desired movements, my description does not produce any particular trajectory. It rather defines the space, in which a null-space controller can optimize an efficient, smooth trajectory. This under-specification is intentional: Some learning methods, for example, prefer the center of their learned tolerance regions. In a new situation, however, it may make more sense to move the robot joints away from its limits or optimize for manipulability or force. My movement description is modular in this sense, too, and can be combined with all of these optimizations.

Also, the proposed constraints lack any notion of dynamics. Thus, they only work for quasi-static settings, which was an acceptable restriction in my experiments. Even for dynamic settings, my description can model the quasi-static part. Specifying e.g. acceleration constraints on the pancake, so that it won't fall during the flipping, would require a completely different execution engine that must plan ahead in time.

The following chapters explain in greater detail the construction of movement descriptions and their execution.

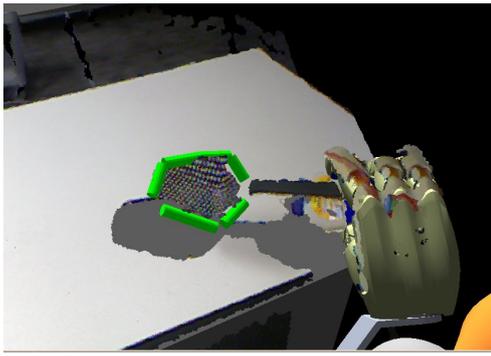
Chapter 3

Generation of movement specifications

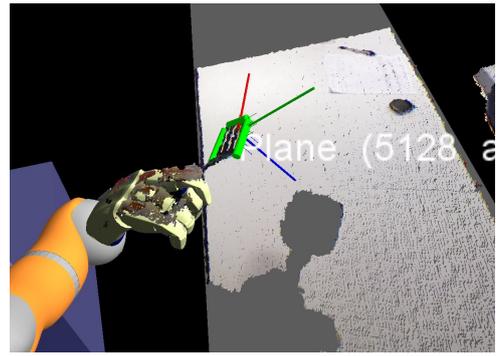
After the general overview given in Chapter 2, I present the methods that were used to perceive geometric features in objects, generate movement specifications and evaluate them on an abstract level. Then I investigate how to specify desired contact forces for every constraint direction. I conclude with the exemplary specification of several every-day movement tasks.

The details of the movement specification, the control framework and null space exploitation are discussed in Chapter 4.

3. Generation of movement specifications



(a) Line features detected on a skimmer



(b) Line and plane features detected on a spatula

Figure 3.1.: *Detected line and plane features on a skimmer and a spatula*

3.1 Geometric Feature Detection

The movement specifications in the previous chapter were referring to parts of tools and objects like corners, edges and faces. In this section, I provide grounding for these entities by specifying detection and localization methods for geometric features in tools and objects. These methods use a 3D camera and a high resolution camera.

In order to enable reasoning about such features, I keep them simple and use only three basic feature types:

- **Points** for corners, center points, etc.
- **Line segments** for edges
- **Plane segments** for faces, blades, etc.

Some of these features are depicted in Figure 3.1. By perceiving objects to such detail, the planner obtains knowledge about the objects' composition and can use this knowledge to 'understand' their function. Not only can the robot distinguish between object parts like handle or blade, but it can deduce the role of every corner and edge in placement and alignment tasks. Such information would be hidden by simply placing a coordinate frame on the object. By adding the detection of concavities and holes, the robot can distinguish between tools like spoon and skimmer and select the right tool for a given task out of a set of unknown items.

The 'classical task' of a localization algorithm is to fit some model to incomplete and noisy sensor data. The origin of the localized model is then passed to the controller to execute the movements. It is not clear, what meaning the x- y- and z-axes have, or if they have a meaning at all, like the x- and y-axes in the cylinder in Figure 3.2).

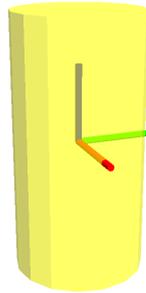


Figure 3.2.: *Cylinder coordinates placed at the center with the z-direction aligned with the cylinder axis. There is one degree of freedom due to the symmetry of the cylinder.*

Describing an object by its task-relevant features, this ambiguity does not occur. The object parts have a unique representation and each number has a specific geometric meaning. For the control-relevant description, a feature is defined by its position and a direction vector. The length of the direction vector defines the length of a line segment or the radius of a plane patch. For points, it can encode the size of a sphere, but the direction itself is unused. These features are so simple that they do not define a coordinate system: A single feature still allows a rotation around its axis of symmetry: At this level of granularity, two features are required to constrain all six degrees of freedom. Just as the selection of a second feature, the placement of a coordinate system is ambiguous: Is the edge of a cube related to the top or side face? This decision is task dependent and is delayed in the proposed framework until the task is specified. Although it was not necessary to complete the kitchen tasks I encountered, defining a complete coordinate system was useful for compatibility with other approaches, e.g. to facilitate testing

A complementary method, described in (Tenorth et al., 2013), is to fit a CAD model and fit geometric shapes into these CAD models. For these approaches, the position and orientation of the model's coordinate frame becomes irrelevant, what counts are the positions and directions of the detected features.

3. Generation of movement specifications

The following subsections describe how geometric features can be used to classify tools, based on an abstract description in the robot knowledge base. After that, I describe how line segments and plane segments as well as concavities and holes can be detected in a tool that the robot has in it's hand. I then give a short description of the approach for analyzing CAD models and relate it to the feature detection. I conclude with an outlook on how a detailed semantic object model can guide CAD model matching and improve precision where the task at hand requires it.

3.1.1 Tool Classification

The following example of tool classification shows how useful it is to understand the parts and features of which an object is comprised.



Figure 3.3.: *The kitchen tools that were used for classification*

The tools depicted in Figure 3.3 are classified based on an ontological description using the features **SharpEdge**, **FlatSurface**, **Concavity** and **Hole**. These features were selected based on the requirements that were found for tools used by TUM-Rosie for food preparation.

Additionally the feature **Handle** is defined, that the robot already has in the hand. The tools which are considered for the classification task can be described using mainly the transitive predicate `properPhysicalParts` taken from the `KNOWROB` system (Tenorth and Beetz, 2009). It expresses, that an object has a physical part and can be read as “has”. Additionally I use a simplification for numeric relations like `min` and `max` to express a more specific quantification than e.g. `exists`. Using these features and predicates, these kitchen tools can be described in the following way:

```
#Definitions:
Class: HandTool
  properPhysicalParts min 1 Handle
Class: Blade
  properPhysicalParts min 1 SharpEdge
  properPhysicalParts min 1 FlatSurface

#Tools used in this paper:
Class: Spatula
  SubClassOf: HandTool
  properPhysicalParts min 1 Blade

Class: Spoon
  SubClassOf: HandTool
  properPhysicalParts min 1 Concavity
  properPhysicalParts max 0 Hole

Class: Skimmer
  SubClassOf: HandTool
  properPhysicalParts min 1 Concavity
  properPhysicalParts min 1 Hole
```

For example, a Skimmer has at least one concavity, at least one hole and (being a HandTool) at least one handle. This ontology allows to classify tools based on the presence of visual features, as is demonstrated in Sections 3.1.3 and 3.1.4.

A side effect of this classification is a semantic labeling of the tool's sub-parts. That, in turn, is the key to grounding instructions like 'align the blade with the oven' in the robots perception. Given that the tool parts 'blade of the spatula' and 'oven' are localized, Section 4.1.2 demonstrates how such instructions can be transformed into a control rule.

The features defined here might be able to express more tools than those three, but there were enough examples to evaluate the method on those three classes.

3.1.2 Feature Detection Scene Setup

Having motivated the perception of geometric features in tools, I now describe how the detection and localization methods are implemented. The perception routines outlined here were developed by Ulrich Klank and are described in more detail in (Kresse et al., 2011). These algorithms were developed for tools that the robot can grab and position and orient to a favorable pose for analysis. This simplifies the

3. Generation of movement specifications

detection process and ensures the best results possible. In particular, I assume that the robot can:

- Grab the tool on its handle
- Move it in front of a uniform, high-contrast background
- Easily segment the part of the tool that sticks out of the robot's hand
- orient the tool so as to maximize the visible area of the tool.

I assume that a suitable grasp can be computed, either using a grasp planner or a heuristic. Taking into account the mean color of the tool (or what was visible of it during grasping), the environment can be searched visually for large areas of a contrasting color. Then the tool can be moved in front of it. After self-filtering the robot out of its 3D camera view, the tool will probably be the only part near the robot hand. This is the tool part of interest, which – thanks to the high-contrast background – can also be segmented on based on the 2D image only. Using this segmentation, the visible area of the tool can be computed. Maximizing this area can be implemented with a simple gradient-descent-like algorithm.

Sensors like the Microsoft Kinect have difficulties with shiny objects, which may not reflect enough of the projected pattern into the camera. Furthermore, the 3D edges are not very precise; Therefore the perception system is augmented with a high-resolution camera for line detection and localization.

In order to extend these perception methods to objects that lack a handle, this perception approach can be generalized to analyzing other objects than hand tools: All that is needed is the generation of a suitable view on the object. If the robot can pick up the object, then the largest-area approach can be used here, although occlusion by the robot hand should be minimized as well. If the robot chooses a suitable grasp, then both measures can be optimized simultaneously. Objects that are hard to pick up for the robot must be viewed from an appropriate direction. However, achieving a uniform and contrast-rich background may be more difficult in this case. The presented possibilities underline the advantages of a robot-mounted camera over a static one. Robots need to exploit these possibilities in order to achieve the high reliability that everyday tasks require of them.

In the following paragraphs, I outline the detection approach: I first fit a plane to the tool in order to detect tools with a blade. I then analyze the distances of the object points to this major plane to detect concavities like in spoons. Finally, I use the 2D camera image to detect holes and to localize edges.

3.1.3 Feature detection and localization

Knowing the approximate size of the tools in question, the robot can derive suitable cropping boxes to select the points of the tool that 'stick out' of the robot's hand.

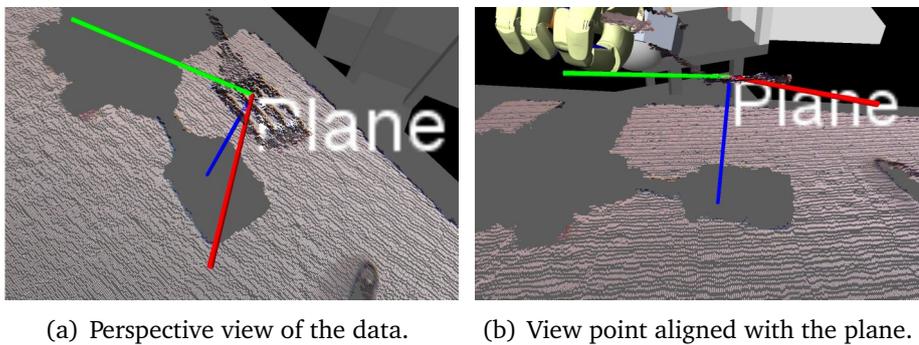


Figure 3.4.: *A point cloud acquired with the Kinect sensor with a coordinate system showing the orientation of the extracted plane. The blue z-Axis represents the normal of the plane.*

After this selection, I perform an SVD on a $(n \times 3)$ matrix that contains all n points as rows. The resulting first Eigenvector is the plane normal. This very simple analysis resulted in a reasonable plane that at least intersected the tool at a relevant position. For planar tools, this plane was also very close to the tool's surface. Imperfections were due to points of the handle that could be discarded with outlier-rejection methods.

For instance, this analysis could be performed in using the height map that is generated for the concavity detection in the next paragraph. This map encodes how far the actual points are from the fitted plane. If the tool is classified as planar, then this distance could be optimized with a robust least-square minimization.

As a next step I distinguish a planar tool like a spatula from a non-planar tool with a concavity – like a spoon. For this analysis I create a low-resolution height map of

3. Generation of movement specifications

the tool like in Figure 3.5. Every pixel of this image encodes the average distance of the points in it's volume to the plane. If the tool has a concavity, then there should be a minimum in the center and a higher ring around so that 'no water can flow out'. I consider a convexity as the back-side view of a concavity: I just flip the sign of the comparison for concavities treat it as if I would have found a concavity on the back side.

In order to find the most probable location and size of a concavity, I place two concentric circles on the height map such that the difference of the mean height under their border is maximized. The circles are shown in Figure 3.6 (a), together with the directions that are searched to optimally place the circles. Because of the low resolution of the height map, this search can be done very fast.

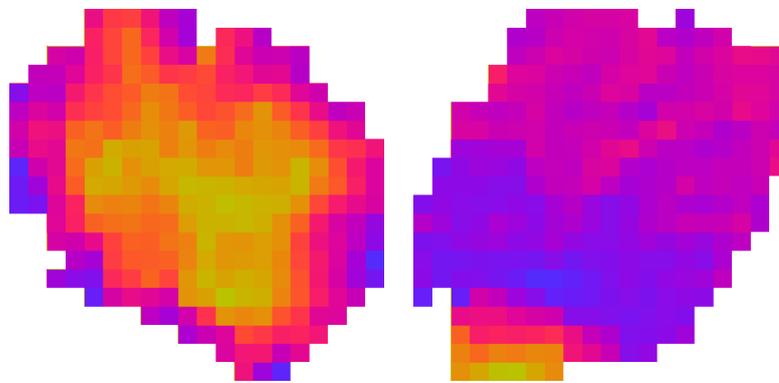
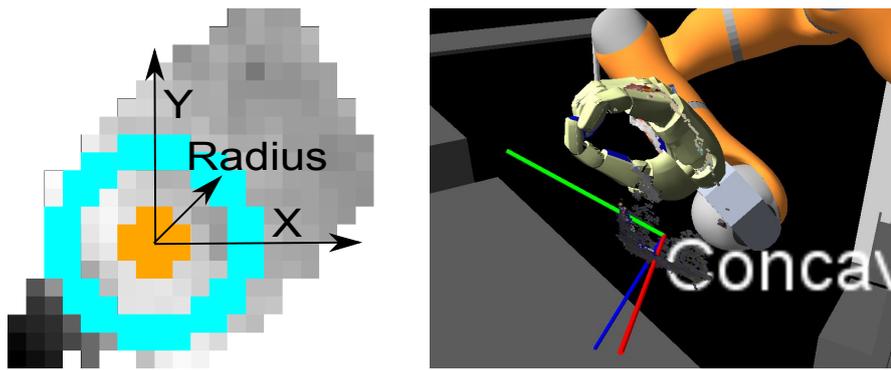


Figure 3.5.: *Height maps of a tool with concavity (a spoon, left) and without (a spatula, right).*

The circles with the biggest height difference are then analyzed whether they mark a real concavity: It must be a 'closed basin'. This is the case if the minimum height on the 'rim' is higher than the maximum height in the center. For a convexity the inner minimum must be higher than the outer maximum. Some outliers are allowed, so long as they are less than the thickness of the outer circle. The position of the concavity in space is on the former plane surface at the center of the circles with the z-Axis pointing into the concavity. An example for this representation can be seen in Figure 3.6 (b).

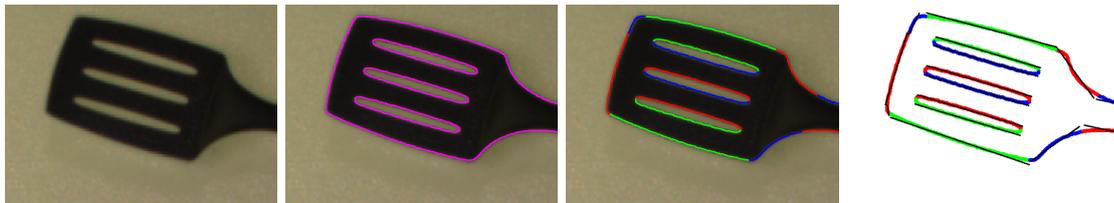
In order to localize sharp edges on the tool, I search for straight lines of a minimum length in the camera image. The extraction process is visualized in Figure 3.7.



(a) The three degrees of freedom for locating the center of the cavity using two concentric circles as a filter.

(b) A large skimmer in the hand of the robot in the Kinect's data with a concavity visualized with a coordinate system.

Figure 3.6.: *Concavity Detection.*



(a) The spatula in the RGB-View.

(b) The edges extracted from the spatula.

(c) Segments of edges.

(d) Fitted lines on the edges.

Figure 3.7.: *The line fitting process.*

For detecting the edges, I first use a standard sub-pixel canny edge filter. This avoids additional forks at intersections. I then employ a split & merge strategy in order to find long and straight edge segments: The edges are split into connected segments. They are split again until some minimum collinearity criterion is fulfilled, that is, the direction along a segment must not vary too much. Then the segments are joined again, if the edge direction is similar and the closest points of the two segments are close to each other. The result of this operation is shown in Figure 3.7 (c). On the resulting segments I pick all those with a minimum length and fit a line to them using a robust least square regression, show as black lines in Figure 3.7 (d).

These lines are intersected with the plane that I extracted in the 3D point cloud by projecting the lines' end points.

3. Generation of movement specifications

Using a morphological closing operation, a difference operation and a connected component analysis I extract holes in the tool.



(a) The holes in a spatula.

(b) The edges (green cylinders) in relation to the plane extracted for this spatula

Figure 3.8.: *Examples for results of hole detection and edge extraction.*

The holes which I detect for the spatula can be seen in Figure 3.8 (a). All lines around holes are discarded. The existence of holes is reported as a result as well as the position and length of the edges in space. Only Edges of a significant length are reported, which are in case of the exemplary spatula the three visible in Figure 3.8 (b).

The detected features with their position and direction are stored in the KNOWROB knowledge base and can be accessed using Prolog queries. Possible queries include the longest edge in an objects of the biggest concavity.

3.1.4 Tool Classification Results

I tested feature detection by classifying the tools from Figure 3.3 according to their features. Each tool was analyzed several times and the results are summarized in Table 3.10. Some detection results are depicted in Figure 3.9. The classification result was very robust. Only the wooden spoon could not be categorized correctly, because it's concavity is so flat that it disappears in the Kinect sensor noise.

Sometimes, additional edges were detected between tool-head and handle, which accounts the different number of detected edges. Big holes are accurately detected, the existence of small holes is also determined reliably. Only the number of small holes varied due to the limited resolution of the camera.

3.1.5 CAD-Model Feature Extraction

In addition to the robot's own cameras, online 3D galleries like Google Warehouse provide a vast resource for 3D object models, that can be searched by key words. It is conceivable, how the aforementioned algorithms can be applied to such models by simulating a high-resolution 3D camera: The object is rendered and the color- and depth buffer are used as the simulated, noise-free input data.

In addition, work by Stefan Profanter and Moritz Tenorth ((Tenorth et al., 2013)) can directly fit primitives to the mesh models in order to detect object symmetries and extract parts like handles or concavities. This approach has the advantage that there is no hidden back side of an object, like for the tool calibration from the previous subsections. However, the technical quality of these models varies greatly and the object dimension is usually unknown. The following paragraphs outline the aforementioned work to deal with these problems and achieve a useful CAD model analysis and -segmentation.

The analysis algorithms were implemented as *computable Prolog properties*. When evaluated, they perform curvature estimation, segmentation and shape fitting on the

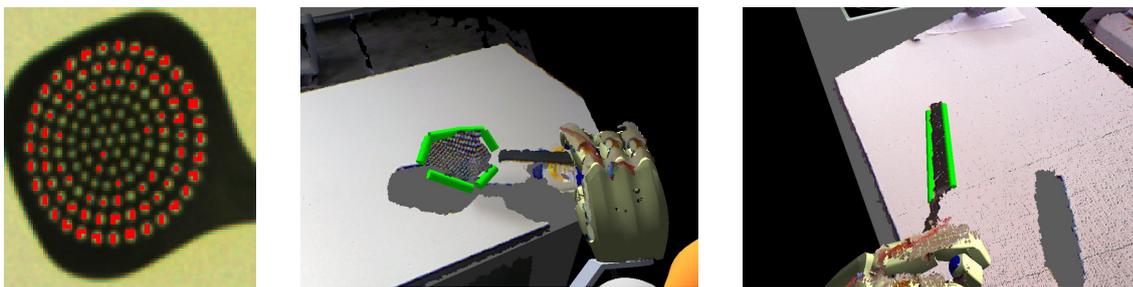


Figure 3.9.: *Visualization of several results: First the small holes which could only be partially detected, then two examples for extracted edges.*

3. Generation of movement specifications

Tool	Concavity/Plane	Num Edges	Num Holes	Res. Tooltype
	Concavity	6	81	Skimmer
	Plane	3	3	Spatula
	Concavity	2-3	0	Spoon
	Concavity	0-1	0	Spoon
	Concavity	4-6	22-35	Skimmer
	Plane	2	0	Spatula
	Plane	0	0	HandTool
	Plane	3-4	15	Spatula

Figure 3.10.: *The visual exploration results for the set of tools, errors are marked in red.*

3D meshes and return the result, as if it had been specified as facts from the start. Prolog is used throughout KNOWROB as an inter-lingua for querying the knowledge base and invoking it's reasoning methods.

If necessary, the vertex normals are computed, using a weighted average of the adjacent triangle normals. Then the curvature is computed per face, using these vertex normals. By inserting them into the 'second fundamental tensor' and solving using a least squares approximation, the per-triangle curvature tensor is obtained. The per-triangle curvature tensors are again averaged to obtain a per-vertex curvature tensor. The weighting scheme, both for the vertex normal and vertex curvature computation is the Voronoi area of the respective triangles. After computing the principal curvatures (The minimum and maximum curvatures at a given vertex), both the mean and product of these two magnitudes is computed. These two values are then used

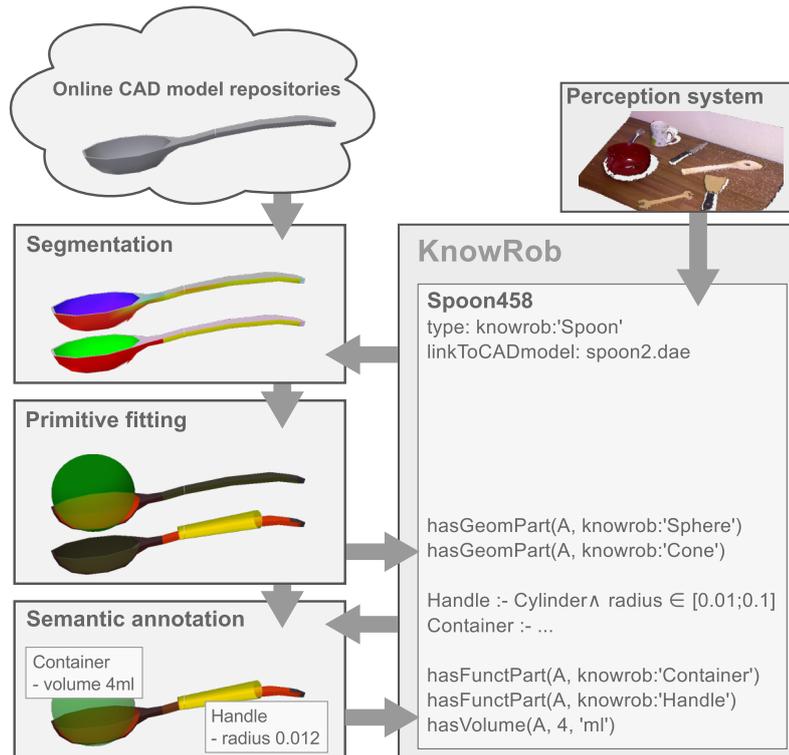


Figure 3.11.: Overview of the CAD model reasoning by (Tenorth et al., 2013)

to create a coloring scheme that is used to distinguish between concave and convex sphere- and cone shapes (or parts thereof). On the recognized parts, the respective spheres and planes are fitted, minimizing a summed quadratic distance between vertices and shape. Cone- and cylinder fitting was made robust against small, differently shaped triangles at the ends. The result is a collection of fitted shapes, consisting of their position, size, volume and concave/convex information. Then, based on these size- and shape properties, containers, handles and supporting planes are identified. These queries are implemented as Prolog queries in the knowledge base, so they are themselves available for reasoning.

This part-based approach for object description can even be beneficial to the CAD-model-matching itself: The features provide better clues as to which part of an object or tool is most important.

Most matching algorithms give all parts of a model equal importance, only heuristics like size and curvature are used for achieving a 'reasonable fit'. With the segmentation into its parts and geometric features identified, the matching quality can be assessed

3. Generation of movement specifications

specifically for the task. For instance, when grabbing a screw driver, the handle must be matched with sufficient precision, so that the grasp is successful. When using the tool, its tip must be found with very high precision and the main axis must be matched up to some degrees. Such a weighting can assess and steer the matching process so that – based on this quality measure – the important features can be matched with more precision.

3.2 Task Planning

The previous chapter has outlined my movement representation- and execution system in general. This architecture puts the responsibility to construct a detailed movement description on the high level planner component.

Traditionally, such a planner only deals with relatively coarse plans and abstract away the detail decisions, which are delegated to a movement execution module where the necessary heuristics are implemented. The proposed architecture deliberately pulls these decisions into the high level planning module, which has access to many more relevant sources of information, both about scene geometry and background knowledge. Here, relevant pieces of information can be combined in an informed manner and their influence on desired and undesired effects can be remembered, simulated, understood and reasoned about. Thus, the planner not only deals with the question *what* to do, but also *how* to do it.

In this chapter, I tackle these sources of information. I consider plan generation from web instructions like ehow.com (Tenorth et al., 2009b, 2010), the robot knowledge base KNOWROB (Tenorth and Beetz, 2009), which contains object models, plan templates, location and purpose of furniture and more, human observations, the perception system, and action-effect relations gathered from experience, naive physics knowledge and simulations (Kunze et al., 2012). I evaluate how they can contribute to a movement representation that is directly executable on the robot controller.

The constraint construction itself is a rather abstract, symbolic process, since it is mostly about selecting geometric features and feature functions. This task is well suited for symbolic high-level planners. The direct grounding in the robot's perception, and the semantic labeling of every movement description component allows to remember the origin of a constraint so it can be adjusted later on. For any adjustment, the intended effect on the movement controller is intuitively clear by the design of the movement description.

A goal of the specification process is to keep a movement specification as flexible as possible: All constraints are given as ranges and only necessary constraints should be specified, so that the robot itself can adapt to new situations.

3. Generation of movement specifications

All these sources of information are human-made, incomplete and possibly contradictory. Therefore, I present a way how the influence of new constraints can quickly and automatically be tested in a reduced simulation environment. This issue will be discussed in Section 3.2.5, where such a simulation is proposed to link constraints with desired and undesired effects.

In the following sections I show how a constraint-based movement description bridges the very abstract and symbolic domain of a task planner to a powerful control architecture. I show how common sense knowledge in the KNOWROB system supports the construction of constraints from incomplete data sources.

The specification process itself can be split into several steps. Specifically, the planner must define the following:

- A partial-order sequence of constraints
- For each constraint:
 - Which two feature to relate to each other
 - Which feature function to use
 - What is the desired range for the value of the feature function

This construction process is very modular: Every constraint and each of these steps can be done using a different source of information.

Most of the robot's sources only contain partial information; however, the constraint-based representation is executable at all times and improves its behaviour with every added (correct) constraint.

In the rest of this section I first argue why 'classical' task planning is so difficult for open-ended scenarios. Then I explore natural language web instruction, the robot's common sense knowledge and human observation data as possible sources for constructing constraints. After that I discuss how desired and undesired effects can be used to judge candidate constraints. Finally, I explain how a constraint-based movement specification induces a fine-grained monitoring which can be used by the planner without further programming effort.

3.2.1 Comparison to 'classical' planning task

'Classically', a (hierarchical) task sequence is generated by a 'planner' component by means of preconditions and effects, down to some degree of abstract actions of the kind 'pick up spatula', (Fikes and Nilsson, 1971; Erol et al., 1994). These actions are then parametrized and performed by an execution module. Usually, for this kind of symbolic task planning, one must carefully specify all the preconditions and effects of all actions that the robot can apply. The involved physics is modeled to a degree that allows to select the right action, by specifying a post-condition that shall hold at the end: The goal of the plan.

Let us consider how a planner might select the right action to 'grab' a pancake on the oven. After pouring the liquid dough onto the oven, this dough has been transformed into a half-baked pancake: The upper side of the pancake is still liquid and delicate, so it should not be touched yet. Therefore, the robot should rule out the actions of grabbing the pancake with its gripper, or using a fork. The oven itself is hot, heavy and attached to a cable, so the planner should rule out throwing the pancake in the air by moving the oven. In a similar manner, the planner must exclude *all* actions that make no sense for this task. This becomes more difficult, if the robot can perform many different actions.

This kind of reasoning usually requires a vast amount of common-sense knowledge, typically in a contradiction-free form so as to allow logical inference from this knowledge base. Another challenge is, that when the pancake first appears on the oven, it is a spot of liquid dough, that is transformed through heat into a solid pancake. Describing that the dough on the oven disappears and – through the effect of heat – turns into a pancake is difficult to describe in today's logical representations.

In (Morgenstern, 2001), Morgenstern has used 66 axioms to formalize the task of egg-cracking. They include simplified descriptions of geometry, containers, liquids, gravity, pouring and more. Despite this amount of work, many questions remained unanswered by that system. Each new task requires more, carefully crafted axioms to describe the effects of the robot's actions. Each added axiom must not only allow to select the right action, but must also avoid any wrong action, especially in all the previously specified plans. In light of this complexity and the amount of expert knowledge that the specification of each new task requires, the use of predefined

3. Generation of movement specifications

plans seems more appealing. Instead of always explaining to the robot *why* it must use a spatula, I can simply tell it to do so. When trading the effort of hand-coding a specific task sequence against the effort of extending the formal planning language in a contradiction-free way, then directly specifying a new task sequence is often easier.

3.2.2 Web instructions

Rather than generating a sequence of actions from an engineered formal world description, the robot obtains existing plans from its plan library or from web instructions.

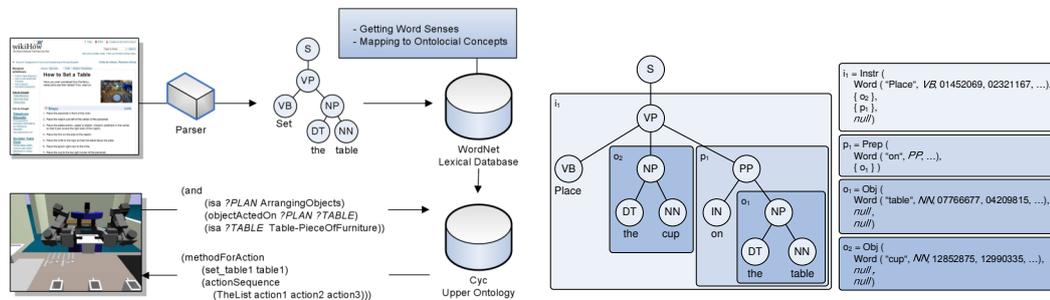


Figure 3.12.: Overview of the web instruction translation (left), Parse tree of the sentence "Place the cup on the table" (middle) and the intermediate data structure that represents this instruction (right) (pictures are from (Tenorth et al., 2010))

On-line web instruction sites like ehow.com or wikihow.com are a remarkable resource of common knowledge: More than 172,000 articles in wikihow.com and more than 2,000,000 articles and videos on ehow.com cover the whole range of everyday activities, including many household tasks. These instructions may be of varying quality, incomplete and contradictory. Their interpretation requires grounding of these instructions into objects in the robot's environment and completing missing instructions.

Nonetheless, the work in (Tenorth et al., 2010) has demonstrated a successful translation into a robot plan. In this work, the instructions are processed with a Probabilistic Context Free Grammar (PCFG) parser, which extracts the structure of the

sentences with objects, quantifiers, prepositions and verbs. The words are then compared to the WordNet lexical database (Fellbaum, 1998) for resolving synonyms and the Cyc ontology (Matuszek et al., 2006) to interpret the words. This information is then converted into a sequence of actions that is stored in the KNOWROB knowledge base (Tenorth and Beetz, 2009), linked to the robots world knowledge. A typical action consists of an action-verb, a preposition and two objects which it relates to each other. These actions are the entities that are later mapped to movement constraints.

Being written for humans, these instruction can assume a fairly large amount of common-sense knowledge, and thus can omit steps that are obvious to the human reader. Actions like switching on the oven (or switching it off) are good candidates for omission – humans know that they are required: When using an oven, they know that turning on the oven produces heat which in turn bakes the meal. These actions need to be added to the translated instructions by the robot’s own world knowledge. The robot knowledge base KNOWROB implements a number of reasoning methods to complete such plans (Tenorth and Beetz, 2012).

Another challenge is that often in these instructions, objects are transformed: Mixing together eggs, milk and flour turns these objects into pancake dough. Through baking, this dough is further transformed into (several) pancakes. This transformation is usually not mentioned explicitly – some objects are used in one instruction which causes the transformation, then the next instruction uses the result. This issue is addressed by building a transformation graph that tracks these transformations and the actions that cause them. Sometimes, human instructions are not precise in that they don’t refer to the right sub-parts of objects: “push the spatula under the pancake” actually means “push the blade of the spatula under the pancake”. Also these flaws must be fixed in some way.

Many aspects of this plan interpretation are outside the scope of this thesis. But the structure of the actions, relating objects with an action verb and preposition align well with the structure of constraints, which also relate two object to each other with a feature function and a desired range that this function should have. In addition to the action sequence, web instructions also contain valuable hints and caveats, sometimes in the form of desired and undesired effects. Such hints should be incorporated into the movement as far as possible.

3.2.3 Robot Knowledge Base and perception

With the coarse structure of a movement retrieved from robot knowledge base resources, the movements must be parametrized and questions like the following must be answered:

- How to align the spatula before touching the oven
- How hard to push down
- How far to push the spatula under the pancake
- How fast to push the spatula under the pancake
- How high to lift

Rather than leaving these details to the movement execution, the planner gathers (some of) these details from the robot knowledge base and the perception results.

The goal is to fill in these movement parameters at a symbolic level and keeping explicit links to the information from which they were derived: For instance, the lifting height might have been computed using the pancake size and spatula size (lower height limit) and oven size (upper height limit). Remembering this link allows to recompute this parameter when the scene changes.

Since some knowledge must be considered unreliable – at least in the context at hand – it is important that a movement can be performed at any incomplete state. This is necessary for a simulation to evaluate any changed or additional constraints. In this situation, it is no longer necessary that formal knowledge is complete and contradiction-free. A mostly correct knowledge base can suggest improvements that can be tested individually. Various depths of movement simulation are discussed in section 3.2.5, where they play an integral part in the construction of new constraints.

Let us consider some information that is available to the robot about the pancake-flipping task. The perception and basic geometric reasoning provide the following facts: Both the oven and the pancake are round. The pancake is a flat object on a flat and horizontal supporting plane – the oven surface. The lower side of the spatula

is in contact with the oven surface (that side is being baked). The spatula has a flat blade, a front- and two side edges and an axis of symmetry. The blade has about the same size as the pancake. This tool allows for three different contacts with a plane: point-plane, line-plane and plane-plane. Some actions depend on this contact situation.

The knowledge base might contain the following information: The spatula is pushed *between* oven and pancake, and thus should be aligned with their contact surface when 'pushing under'. After that, the spatula becomes the supporting plane of the pancake. Knowing the objects on a tilted supporting plane can slide down (naive physics knowledge), the spatula should be held *horizontal*. However, this sliding down is the intention during the turning movement, at the end of the flipping task. The *lifting height* must be high enough so that the turning works without a collision (implicit undesired effect). After the turning, the pancake falls down vertically (naive physics knowledge). Therefore, in that moment the spatula should be above the oven.

The planner has a fine number of geometric details to ground it's reasoning. But even simple movement details are difficult to infer in a general way. But also natural-language hints that may have been gathered from web instructions, can be exploited: 'keep the spatula horizontal', 'push firmly', or 'point the spatula towards the pancake' are hints that are possible in (admittedly overly verbose) instructions. Still, that would be natural description on *how* a movement should be performed.

In the pancake-example, I assume that the robot has collected the constraints in Table 3.1, which are used by four movement phases. Some of these 'high-level' con-

touch-down = $\{c_p c_a c_n\}$	c_p = point-towards(spatula, oven)
push-under = $\{c_p c_a c_u\}$	c_a = align-for-push(spatula, pancake)
lift = $\{c_h c_o c_l\}$	c_h = keep-horizontal(spatula)
turn = $\{c_o c_t\}$	c_n = move-next-to(spatula, pancake)
	c_u = move-under(spatula, pancake)
	c_l = lift(spatula)
	c_o = keep-over(spatula, oven)
	c_t = turn(spatula)

Table 3.1.: Constraint-based description of pancake flipping (left: movement phases, right: abstract constraints).

3. Generation of movement specifications

straints have two arguments and relate two objects to each other, but keep-horizontal, lift and turn only have one argument. In all of these, the world-feature gravity is involved.

Another distinction is less visible syntactically, but is measurable during execution: Some constraints must be fulfilled during the complete movement phase, like horizontal - others, like move-under imply a movement and can only be fulfilled at the end of the phase. While the first kind reduces the space of possible movements, the second constraint encodes a goal and leaves the smooth trajectory generation up to the movement execution.

The next step is to turn these high-level constraints for flipping a pancake into elementary 1D-constraints. This includes selecting the right features and the right feature-function. Often, the respective center point feature can be used, or the most important plane feature. But sometimes, a more precise definition must be inferred. This is reflected in imprecise natural-language statements: 'Align the spatula blade with the oven surface'. When enumerating the known geometric features, the robot finds the round oven surface and the spatula blade, front-edge and side-edge. Should it align the blade normal with the oven surface, creating a plane contact? Or is it better to align the front- or side edge and create a line contact? In fact, it is more robust to align the front edge and keep a small angle between side edge and oven surface. This question can be phrased as a very precise natural-language-question, whose answer is 'align the spatula front edge with the oven' and 'keep a small angle between spatula side (edge) and oven'. This question can also be resolved by querying human movement data or simulation trials. These are discussed in the next sections.

As can be seen in Table 3.2, some high-level constraint functions have a 1-to-1 relation to feature functions. Others are the combination of several elementary constraints. This mapping is stored in the KNOWROB knowledge base in the form of *movement patterns*, as described in (Tenorth et al., 2014). In these patterns, the features that are related to each other can be described with the full power of ontological knowledge and the inference engine of KNOWROB: The main axis of an object or its center are easily described in logical terms. It is remarkable, how each 'high-level' constraint resembles a natural-language statement that *could* be found in a verbose explanation and its translation an even more pedantic but still understandable version of that constraint. Part of this translation is the definition of desired ranges. This

point-towards(spatula, oven)	→	pointing-at(spatula _{forward} , oven _{center}) = 0
align-for-push	→	perpendicular(spatula _{front-edge} , oven _{normal}) = 0 ∧ perpendicular(spatula _{side-edge} , oven _{normal}) < ε
move-under(spatula, pancake)	→	height(spatula _{center} , oven _{center}) = 0 ∧ distance(spatula _{center} , pancake _{center}) = 0
move-over(spatula, pancake)	→	height(spatula _{center} , oven _{center}) > 0
move-next-to(spatula, pancake)	→	height(spatula _{center} , oven _{center}) = 0 ∧ distance(spatula _{center} , pancake _{center}) = r _{spatula} + r _{pancake}
keep-horizontal(spatula)	→	align(spatula _{blade-normal} , gravity) = 1
lift(spatula, oven)	→	height(spatula _{center} , oven _{center}) > r _{spatula}
turn(spatula)	→	align(spatula _{blade-normal} , gravity) = -1

Table 3.2.: Translation from high-level constraint to elementary constraints

information is implicitly encoded in the high-level constraint statements, hiding only some qualitative understanding of surface normals, and the terms perpendicular and aligned.

For some constraints the desired value is simply 0 or 1, for others it depends on the size of the objects or features. For example, placing the spatula blade next to the pancake needs the size of the pancake and the size of the blade in order to compute the desired distance between those two objects. Rules like this are encoded implicitly in prepositions like 'next to' and the necessary dimensions are delivered by the perception system and are encoded in the feature sizes. Thus, a useful heuristics for encoding such a distance r is:

$$r = \alpha \|dir(f_1)\| + \beta \|dir(f_2)\| + \gamma \quad (3.1)$$

with $\alpha, \beta, \gamma \in \{-1, 0, 1\}$, where $dir(f)$ is the direction vector of the feature f . This allows, for instance, to express 'next to' as $(\alpha = 1, \beta = 1, \gamma = 0)$. This would place the spatula right next to the pancake. The task however, does not require the spatula this close to the pancake when touching the oven surface. This is actually a minimum distance – a smaller distance would be interpreted as 'on top of'. In practice, humans add a padding to objects, depending on their size and fragility and the human precision.

3. Generation of movement specifications

This is a good example, where language can define a constraint very precisely: By also knowing the terms 'on top of' and 'overlapping', a very sharp definition of 'next to' can be made. The bigger the robot's vocabulary, the better it's interpretation of instructions.

If it is an equality constraint, the planner must decide, how precisely the goal must be hit. Although the wish for perfectly precise control can be expressed, a controller will be limited by its actuators. Furthermore, the movement monitoring needs a threshold to declare a constraint as fulfilled (and therefore, the next movement step can be started. The size of such a goal range i.e. the precision with which the constraint must be fulfilled depends on the task and is difficult to state in general. The precision with which the task *can* be fulfilled depends – amongst others – on the precision of the perception; a detail that may be available from a modern perception system.

Sometimes, limits for the same task function are specified several times independently. To combine them, one can use interval arithmetic to compute the – hopefully non-empty – range of desired values. I try to keep these ranges as big as possible, so that a movement specification captures as many valid trajectories as possible. Another approach to this problem is taken in (Kaelbling and Lozano-Pérez, 2011): A heuristic is used to choose a particular value from a given range. If this value is in conflict with a constraint that is specified later, then backtracking is used.

Independent of hints from the current instruction, there is a prevalence for certain constraints with certain tools or objects. For example, aligning a flat blade with a working surface, or aligning a ladle horizontally is a common constraint that often influences the success of a movement. Such constraints can be stored in the knowledge base along with each object class.

In addition, prominent object- and tool features can be selected randomly for a constraint. Candidates may be the largest planes, lines or axes of symmetry. In order to verify these candidate constraints, the task success must be measurable, i.e. desired and undesired effects must be specified and perceived, at least in simulation. The set up of a simulation that checks such a candidate constraint is described in section 3.2.5.

The pancake flipping example reveals, that only very few constraints can be expected to be inferred from coarse, incomplete natural language instructions. Too much is

considered common sense knowledge which, although intuitive, is hardly ever written down explicitly. But those constraint that could be inferred required very little transfer effort. Conversely, this section has shown that almost all constraints have a clear and understandable natural language representation. Stating a fact more precisely in the form of logical constraint is directly reflected in the corresponding natural language statement. This insight is encouraging when considering a dialog with humans: The proposed movement representation not only allows the robot to understand hints at various levels of detail, but it also can explain it's movements and ask for clarification about some aspects.

3.2.4 Human observation

In this section, I examine human movement data of every-day activities as a source for generating movement constraints. In (Beetz et al., 2010), human movement trajectory data is augmented with location annotations and environment sensors that track object movements. This data is segmented using conditional random fields. The segments are then analyzed using Bayesian logic networks, which recognize complex activities in these data. Another approach is presented in (Nyga et al., 2011), which performs clustering on the trajectory data and then uses 'Grounded Action Models' (v. Hoyningen-Huene et al., 2007) to relate those clusters to the context in which they were observed. Since coarse action descriptions are used to recognize an activity, these methods can not be used to discover these same actions sequences. But it is possible to recall all trajectories that correspond to a given action. In order to evaluate the opportunities for constraint generation, a simple experiment of cucumber cutting was set up. Compared to the kitchen data set (Tenorth et al., 2009a) that was used in the mentioned papers, this experiment contains more accurate data at a much faster sampling frequency. This way, I hope to capture some movement details that can be translated into constraints.

The knife was tracked with a Hydra magnetic tracker at 250Hz. The cutting board, cucumber and the knife itself were measured with a combination of hot-spot calibration (Tuceryan et al., 1995; Fuhrmann et al., 2001) and 'teaching' corner points. As a result the following features are known (Figure 3.13 (b)): The tip, upper-edge and

3. Generation of movement specifications

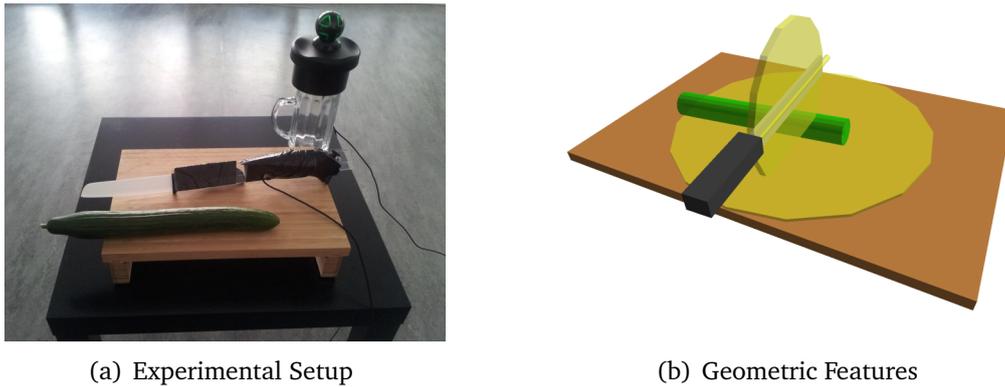


Figure 3.13.: *Cucumber cutting experiment*

normal vector of the knife's blade, the cutting board and the length and pose of the (uncut) cucumber.

Then, all these tool- and object features are collected and combined with feature functions perpendicular, distance and height, which compute the cosine of the angle between two features and the distance projected onto one feature or perpendicular to it (see also Section 4.1.2. The values of these feature functions are a coordinate transformation into a task-relevant space that abstracts away the robot posture and the scene pose in the world. These trajectories in feature-function space are then tested for their movement-energy during the action in question.

Another ranking of probable constraint importance can be made based on feature types like center point, tip, surface or line, or based on the feature function. The 'right' ranking would be the probability with which a particular feature function / feature-type combination occurs in a movement. Given a large corpus of existing feature-constraint-based movements, this probability could be computed directly.

Good candidate 'constraints' are those that change a lot – in the cucumber cutting example this is the vertical movement of the knife, moving from the top of the cucumber down to the cutting board and back up.

Table 3.3 summarizes the relative motion energy in all combinations of tool-feature, object-feature and feature function.

Due to different units for translational and rotational constraints, the rotational energies are all on top of the list. Constraints number 3 and 4 are redundant, according

01	1.000000	perpendicular(blade_{upper-edge}, board)*
02	0.418558	perpendicular(blade, board)*
03	0.251780	perpendicular(blade _{upper-edge} , cucumber)*
04	0.138834	perpendicular(blade, cucumber)
05	0.050511	height(blade _{tip} , board)*
06	0.045091	height(blade_{upper-edge}, board)*
07	0.045091	height(blade, board)
08	0.020111	distance(blade _{upper-edge} , cucumber)*
09	0.020111	distance(blade, cucumber)
10	0.018641	distance(blade _{tip} , board)
11	0.018190	distance(blade _{tip} , cucumber)
12	0.017780	height(blade _{tip} , cucumber)
13	0.015722	height(blade _{upper-edge} , cucumber)
14	0.015722	height(blade, cucumber)
15	0.015363	distance(blade _{upper-edge} , board)
16	0.015363	distance(blade, board)
17	0.000000	perpendicular(blade _{tip} , cucumber)
18	0.000000	perpendicular(blade _{tip} , board)

Table 3.3.: *Relative movement energy in the candidate constraint direction. The constraints that I chose manually are printed in bold, those that were automatically selected after constraint checking have an asterisk attached.*

to the conflict check, as are constraints 8 and 9. Given these facts, this simplistic test results in a good guess about what task directions are important. The constraint 14, which captures the slow movement from one end of the cucumber to the other, was not found as there was not enough movement energy. However, many constraints are fulfilled during a movement where *no* movement energy is detected. Some of these are equally important for the task success. It helps to remember, that human movement control aims for precision only in the directions that are important for a task, those that are unimportant are less of a concern. This is especially true at the very *moment* that a constraint becomes important: While it is irrelevant whether the knife’s blade is vertical during the up-move, it becomes important right before the cutting-down movement. A proper temporal alignment of multiple trajectories can reveal such moments, as has been shown in (Calinon et al., 2007).

When it is clear, which feature function shall be considered on which features, then the trajectories easily yield the values that these feature functions have taken during the demonstrations. These parameters can be taken directly from the movement

3. Generation of movement specifications

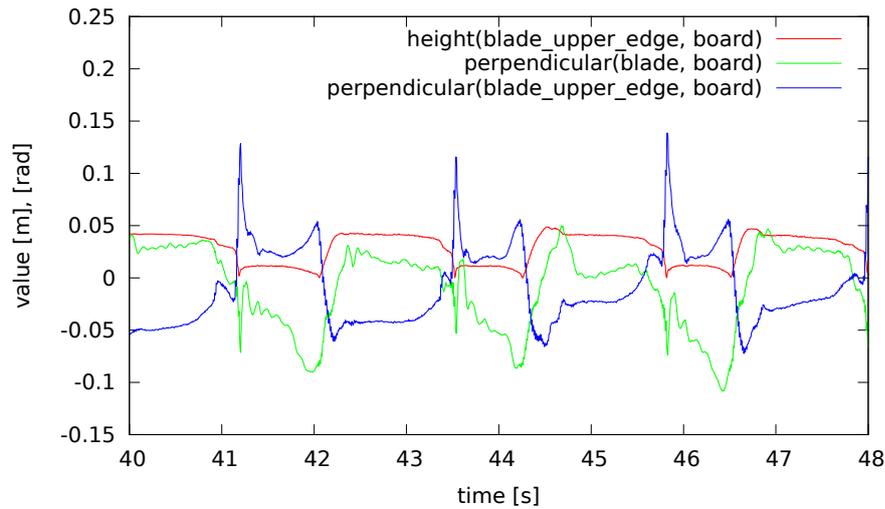


Figure 3.14.: *Cucumber cutting – constraint function trajectories*

data. However, this assumes a very fine segmentation of the trajectory data. In this experiment, the knife was rolled during the cutting movement. But after the cut it was also turned sideways in order to put down the new slice. Nonetheless, the height function in this graph is an excellent candidate for an easy segmentation.

This yields a more or less convenient source for the last 'magic' numbers in the movement description: The geometric features are localized using perception routines, the feature function is represented as a symbol and the desired ranges can be taken from human observation.

The presented procedure also offers a way to convert 'conventional' relative-pose based movements into feature-function based constraints: After constructing a task space with six dimensions, the feature functions are used to convert from 3d Cartesian space into the task space. Then, constraints can be relaxed or removed, one at a time, checking the success rate using simulation or real-world experiments. This allows the robot programmer to convert his programs movement by movement and improving the success rate in small steps.

The transformation into feature-function coordinates is comparable to the dimensionality reduction, like principal component analysis that is sometimes done as a first step in imitation learning. Imitation learning requires a few arbitrary dimensions with high variance and reproduces the movement in these directions. Constraint-

based task programming additionally tries to assign semantics to these movement directions. In addition, movement directions with particularly low variance can be found that otherwise would be lost in the 'remaining' dimensions that are ignored.

3.2.5 Parametrizing movements from desired effects through simulation

After discussing natural language-like knowledge and human movement data, this section investigates a third source of information for constructing constraints: Desired and undesired effects. Often, this problem formulation leads to a quite open reinforcement-learning task, where a reward is given after a long task sequence. For this task, powerful, general-purpose algorithms have been developed. Here, I propose to refine the desired and undesired effects to become monitoring functions that are observable rather immediately and break the learning problem down into finding and parametrizing constraints that yield such a desired effect. For this approach the robot needs the desired effects and a (simulated) execution environment that can reproduce and perceive these effects.

Let us return to a coarse symbolic representation of pancake flipping. The desired effect is to have a pancake that is baked on both sides, undesired effects are that the pancake is destroyed, on the table or on the floor. This action can be split into the phases

1. pour dough on oven (effect: dough on oven)
2. flip pancake (effect: turned upside-down, on oven)
3. move pancake onto plate (effect: pancake on plate)

The desired effect of pouring dough on the baker are to have some amount of it on the baker, the effect of flipping the pancake should be that it is turned upside-down on the oven, and the effect of moving the pancake onto a plate is simply a change of it's location. Undesired effects throughout these action are still that the pancake (or, in the first step: the dough) is on the table, the floor or that it gets destroyed.

In the presented setting, the flipping action, can be further split up into

3. Generation of movement specifications

- move spatula under pancake (effect: spatula above oven and pancake above spatula)
- lift pancake (effect: pancake on spatula (rather than falling back onto the oven))
- turn pancake (effect: pancake falls down, with the former upper side down)
- (shake spatula until pancake falls down (effect: force the falling down, if it didn't occur before.))

As can be seen, the robot task not only naturally decomposes into a hierarchy of actions, but each action is annotated with some desired and undesired effects, where each level has more concrete actions and more concrete effects.

It is noteworthy, that the quality of the desired effects changes with the granularity of the movement description: The coarsest level defines the end result but in order to check the result, further knowledge is required: How does a baked pancake look like compared to an unbaked pancake? How can the robot inspect the other side? At a medium granularity, it becomes feasible to specify percepts that can check the result of each step. The finest granularity contains details about object- and tool poses. It suggests percepts, that can be monitored continuously, and that have causal relationships to simple movement parameters. For example, 'don't let the pancake fall during lifting' relates to a movement constraint 'keep spatula horizontal'. It is these causal relationships that I want to discover.

Having defined the desired and undesired effects at a fine granularity, the next step is to derive perception routines for them. Some of these routines may be challenging in reality due to e.g. occlusion or reflective materials. But a simulation environment has the advantage that all object poses and velocities are readily available. Even though the simulated friction and contact forces may not always be correct, a simulation can give valuable qualitative insights. Work that integrates simulation in the robot's reasoning and decision process has been done by (Kunze et al., 2011, 2012): The simulator Gazebo has been used to recreate challenging scenarios like breaking an egg mixing of liquids. Another light-weight integration of the bullet physics engine and OpenGL based perspective taking is done in (Mösenlechner and Beetz, 2011). Both system can construct a simulation scene from logical statements, run the sim-

ulation and ask logical queries about the result. The following paragraphs propose to use these facilities to evaluate movement constraints. Since the constraints are directly defined on objects and tools, the simulation setup is simple in that it does not require a robot.

A concrete supervision task that is continuous and differentiable, directly induces a movement controller that improves the tasks score as fast as possible.

For example, the lifting of the pancake depends on the spatula being (more or less) horizontal. If the spatula is too steep, the pancake will slide down. This can be perceived (and corrected) in the instant that the pancake starts moving with respect to the spatula. A physics simulator offers a direct way to implement such a controller by evaluating all possible movements in every control cycle:

The controller moves the tool pose in all Cartesian directions and simulates a single time step (if necessary, several simulation steps can be done, but this increases the computational load). Each of these simulations yields a value of the monitoring function. These values are combined into the gradient of the monitoring function w.r.t. changes in the tool pose. Together with the desired value, this gradient is fed to the controller. This computation must be done in every control cycle.

This yields a controller that is directly based on the perception routine, which was implemented for monitoring, in the example, the 'pancake-does-not-move' constraint. However, this control scheme depends on a very fast physics simulation and is a black box that does not offer further insights into the structure of the required movement.

Instead, a simulation can be used to compare geometric constraints with respect to a monitoring function:

Each candidate constraint is added to the existing set of constraints. Then, the simulation is run for one time step, with the added constraint. Again, the monitoring is evaluated and the constraint with the best performance is chosen.

However, this comparison is difficult since it depends on the dynamics of the controller. In order to eliminate this influence, the computed movement of the constraints must be normalized. But this requires a way to combine rotational and translational velocities. Task functions, which represent distances or angles can be normalized in

3. Generation of movement specifications

task-space, and can then be compared with their kind. Other task functions need a heuristic. For instance, a movement could be weighted by the (fastest) linear movement that it causes at the corner points of the tool.

In the spatula-horizontal example from above, the appropriate constraint (the blade normal aligned with gravity) should have the best performance. In special cases, constraining the angle between the spatula side edges and gravity may yield the same result; it is therefore important run these tests for several different situations.

Compared to the simulation-based controller, the geometric constraint is much faster to compute and also more robust: When static friction is simulated, then all orientations where the pancake 'sticks' to the spatula are acceptable: Only for orientations outside that friction cone, the simulation can compute a gradient. The geometric constraint, on the other hand, can always compute a gradient and can be used for control, even when the pancake is still sticking.

Another, very direct approach is to generate random situations in simulation and evaluate the monitoring functions and all candidate constraint. The task is then to find the hypercube that contains all 'good' poses, according to the monitoring functions. But the dimension of the search space grows with the number of candidate constraints, so even a moderate number of them leads to very high computational effort to cover that space. The advantage of this approach is, that no desired constraint ranges need to be selected during constraint discovery. They can be selected later during the evaluation of the results.

In case that only success/failure information is available at the end of a long simulation, the specification task essentially becomes a reinforcement learning problem. If it is expected that only a few (additional) constraints are needed, then these candidates can be added to a simulation run, one at a time. For more general reinforcement learning algorithms, a well chosen, non-conflicting set of constraints can serve as a task-specific coordinate system. Like for the human trajectory analysis, the resulting trajectories can still be interpreted through segmentation and range determination.

It was noted already that it is important to test several situations when validating new constraints. In addition, it is possible to create evaluation examples that are good considering the currently defined constraints, but fail in simulation according to the monitoring functions (or vice versa):

This can be achieved by setting the desired range of a constraint to a specific value and re-running the simulation. If that values was inside the desired range and the simulation fails, then a new starting point to try extra constraints is found. Conversely, if that value was outside the desired range and the simulation still succeeds, then it may be possible to extend that range.

The discovery strategies described in this section are grounded in the desired and undesired effects and depend on their short-term observability in simulation. These strategies may seem brute-force at first sight, but due to a small number of relevant features and feature functions, the search space is limited. It is also possible to rank candidate constraints according to their prior probability. In addition, constraints that are in conflict with an existing constraint set can be excluded right away.

The advantages of this approach are that simulations can be limited to the participating objects, and depending on the monitoring functions, only very short periods need to be simulated. Thanks to the fine-grained movement representation, small aspects of a task can be examined easily. The object-centric description leads to robot-independent results that are accessible for reasoning. compared to reinforcement-learning, the results are easily interpreted in a symbolic way, since they *are* symbolic relations between objects. Due to this grounding, they also are easily adapted to changed situations. For every desired and undesired effect, a specific (set of) constraints is found, which reduces the space of possible movements.

3.2.6 Monitoring

On a real robot, a movement controller must not only execute the commands that are sent from the high-level, but it also must give feedback about it's progress. Through the desired ranges, the movement specification also provides a uniform interface for such a monitoring: The robot simply computes the value of the task functions and compares them to the desired ranges. Without further effort, every constraint is automatically paired with a predicate that tells whether a commanded constraint is fulfilled or violated. This naturally complements the movement construction at the planner side and yields a task-related sensor interpretation, almost 'for free'. This data is focused on the current task and has the potential to replace some of the usual

3. Generation of movement specifications

sensor-interpretation code. With this infrastructure, the planner can answer questions like:

- Which constraint was fulfilled last?
- Were there constraints that were fulfilled and then violated later?

When comparing different executions, some more interesting questions can be answered immediately:

- Which constraints were satisfied faster / slower?
- Was a constraint violated that was fulfilled in a previous execution?

In case of a failure, the robot can produce explanations like: “I could not flip the pancake this time because I could not lift it high enough (and the oven was on a higher table)”.

3.3 Planning Force-Controlled Movements

The previous sections described how relevant movement directions (elementary function), and position ranges (desired values of these functions) could be defined. This is sufficient information to execute movements in free space. This section proceeds to derive constraints on forces that the robot shall exert to its environment. This is important to specify when in contact, that is, during manipulation (intentional contact) and when a collision happens (unintentional contact). Specifying and limiting contact forces avoids damage to the manipulated objects, the environment, or the robot itself. It is also important to maintain a desired contact or to cause movements like pushing a button.

Just as for the desired position ranges, I specify a desired force range. Other approaches specify a desired force (force control) or a stiffness (impedance control). However, having a desired force is harder to check (due to hardware limitations, a desired precision must be specified implicitly). Specifying a stiffness may be intuitive for a programmer, but the applied force can only indirectly be computed. While it is not hard to experimentally find that an egg should be handled with an impedance of 200 N/m with a particular robot, the statement that no more than 10N of force should be applied to an egg is more object-centric and independent of a particular robot and situation. But, for example, the minimum gripping force depends on the friction coefficient of the robot's gripper and cannot be made independent of the robot. Having identified a case of a minimum force (gripping force) and maximum force (holding egg) with related percepts (dropping the object and breaking the egg), a desired force range appears to be a useful choice.

The specification that is decided during planning of a movement is how much force can and should be exerted against the environment in the relevant directions, much like the position range that was derived in the previous chapters.

Due to the inherent coupling of position and force, a robot cannot always achieve arbitrary force- and position combinations. Rather, when a force task shall be specified, the position range serves as a safeguard and vice versa, for a position task the allowed force range avoids to destroy the environment.

3. Generation of movement specifications



Figure 3.15.: *relevant torque directions for maintaining contacts.*

I will now consider how to derive desired force ranges for three important situations:

- avoiding damage at a potential crash
- manipulation like pushing a button
- alignment → contact force and zero-torque constraints

The first case is for free-space movement in an environment, where an unexpected obstacle may obstruct the robot's path at any time. This can be a human who is running into the robot or an object that was hidden from the robot's perception. In this case, the desired external force is clearly zero and the maximum force in any direction is some generic value that does not destroy most objects in the environment or may hurt a human. One might be tempted to set this value very low, but this would hinder the robot to maintain its position constraints or even overcome (the un-modeled part of) its internal friction. This will become more evident in section 4.3.3, where the translation of force- and position ranges into a stiffness and position is discussed.

The second case is – strictly speaking – always intended to cause movement: Whether it is pushing a button down to its contact position, pushing a knife through a cucumber, or pulling a handle to open a cabinet; even triggering a force-sensor button means to cause a microscopic movement which is detected by a strain gauge. The difference to 'normal' movements is often, that a characteristic force is needed for these tasks: Pushing an elevator button requires a different amount of force than cutting a cucumber, cutting a carrot or picking up a slice of cheese for making a sandwich. These forces are associated with the main movement direction and can most often be

learned using other task-related percepts: An elevator button usually lights up when it was pressed; After a successful cut through a vegetable, the knife has moved downwards to the cutting board and a slice of cheese can be detected when it is attached to the fork in mid-air. The result of this learning process is a minimum force that is required for successful task execution. This observation again underlines the importance of constantly supervising all robot actions and perceiving all relevant scene changes.

Finally, force (and compliance) can be used to achieve contact constraints that cannot be perceived precisely enough through vision. By pushing a tool into an object, a remaining gap in the tool position can be closed. By monitoring the force in this direction, that contact formation can be detected. Using such monitoring to stop a movement in contact direction has been successfully used in several demonstrations like pancake baking, sausage cooking and sandwich making. Such a contact force should be maintained in order to avoid breaking contact under perturbations. Furthermore, it provides a direct and fast way to track the exact object surface. This is sufficient for point-like contacts. However, for line- and plane-like contacts, it is further desired to achieve precise alignment. When a contact force is applied to a mis-aligned tool, then a torque is produced, leading to a rotation that improves the alignment.

For this task, compliance can be exploited further: By setting one or two zero-torque constraints perpendicular to the contact normal, the alignment can be achieved with much less force.

When a line contact is desired, but instead only a corner is in contact with the object surface, then the line feature serves as a lever to the contact force which is applied in the middle of the line. The longer this lever is, the easier is it to control towards alignment. A similar argument can be made for a desired plane-plane contact.

The success of this strategy depends on the sensitivity of the torque sensors and the performance of the underlying force control. The less remaining friction is disturbing the controller, the better the alignment result. The zero-torque rotation axis is placed through the middle of the line- or plane feature and aligned perpendicular to the feature-to-be-in-contact:

3. Generation of movement specifications

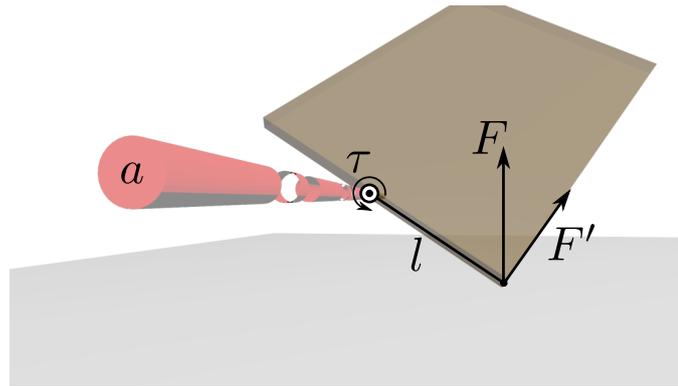


Figure 3.16.: *Lever between contact point and zero-torque rotation axis. The resulting torque is the cross product of the force F and the lever l to the rotation axis α .*

For a plane-line contact, the axis is perpendicular to the plane normal and the line direction. For a plane-plane contact, the axis should be perpendicular to both plane normals. Of course this is highly unstable when the planes are almost parallel. However, it is sufficient to choose axes that are independent and in the contact plane – any such specification will have the same effect for the controller, as will be detailed in section 4.3.4.

Although the success of this strategy depends on the size of the tool and a low robot-internal friction, I show in section 4.3.6.1, that even for a lever of only 0.04m these constraints improve the alignment considerably.

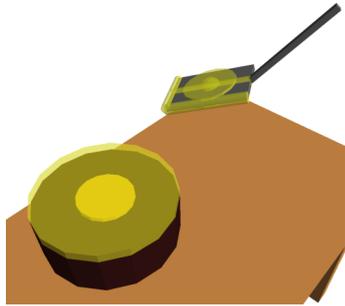
All these force constraints involve positions in some way, at least as a safeguard to avoid that the robot 'runs away'. Thus, impedance control is a natural candidate for implementation.

3.4 Examples for Constraint-based Task Descriptions

In this section, some example tasks are specified using constraints.

3.4.0.1 Pancake flipping

As the running example for in this thesis, almost all aspects have already been explained. The features used in the description are:

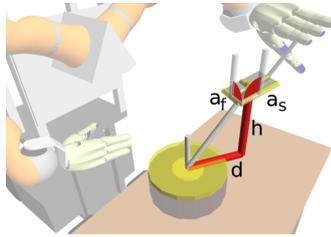


spatula: front-edge
side-edge
center
blade
forward

oven: surface

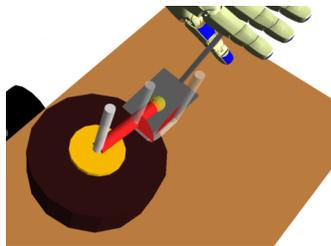
Figure 3.17.: *pancake flipping features*

The following constraints are spanned:



- (p) pointing-at(spatula_{forward}, oven_{center})
- (a_f) perpendicular(spatula_{front-edge}, oven_{normal})
- (a_s) perpendicular(spatula_{side-edge}, oven_{normal})
- (h) height(spatula_{center}, oven_{center})
- (d) distance(spatula_{center}, pancake_{center})
- (o) align(spatula_{blade-normal}, gravity)

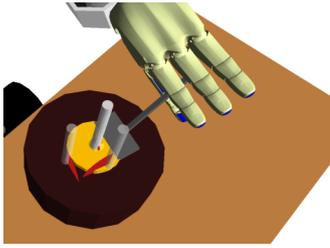
These constraints are used in four phases:



- (p) pointing-at(spatula_{forward}, oven_{center}) = 0
- (a_f) perpendicular(spatula_{front-edge}, oven_{normal}) = 0
- (a_s) perpendicular(spatula_{side-edge}, oven_{normal}) < ϵ
- (h) height(spatula_{center}, oven_{center}) = 0
- (h) height_{force}(spatula_{center}, oven_{center}) = $-F_{\text{contact}}$
- (d) distance(spatula_{center}, pancake_{center}) = $r_{\text{pancake}} + r_{\text{blade}}$

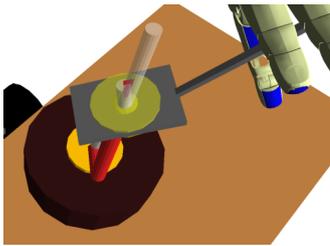
Figure 3.18.: *touch-down phase*

3. Generation of movement specifications



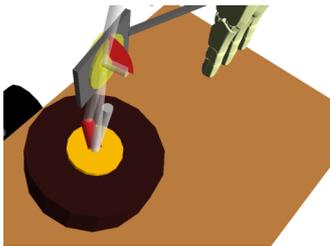
- (p) $\text{pointing-at}(\text{spatula}_{\text{forward}}, \text{oven}_{\text{center}}) = 0$
- (a_f) $\text{perpendicular}(\text{spatula}_{\text{front-edge}}, \text{oven}_{\text{normal}}) = 0$
- (a_s) $\text{perpendicular}(\text{spatula}_{\text{side-edge}}, \text{oven}_{\text{normal}}) < \epsilon$
- (h) $\text{height}(\text{spatula}_{\text{center}}, \text{oven}_{\text{center}}) = 0$
- (h) $\text{height}_{\text{force}}(\text{spatula}_{\text{center}}, \text{oven}_{\text{center}}) = -F_{\text{contact}}$
- (d) $\text{distance}(\text{spatula}_{\text{center}}, \text{pancake}_{\text{center}}) = 0$

Figure 3.19.: *move-under phase*



- (h) $\text{height}(\text{spatula}_{\text{center}}, \text{oven}_{\text{center}}) = r_{\text{blade}}$
- (d) $\text{distance}(\text{spatula}_{\text{center}}, \text{pancake}_{\text{center}}) = r_{\text{oven}}$
- (o) $\text{align}(\text{spatula}_{\text{blade-normal}}, \text{gravity}) = 1$

Figure 3.20.: *lift phase*



- (h) $\text{height}(\text{spatula}_{\text{center}}, \text{oven}_{\text{center}}) = r_{\text{blade}}$
- (d) $\text{distance}(\text{spatula}_{\text{center}}, \text{pancake}_{\text{center}}) = r_{\text{oven}}$
- (o) $\text{align}(\text{spatula}_{\text{blade-normal}}, \text{gravity}) = -1$

Figure 3.21.: *turn phase*

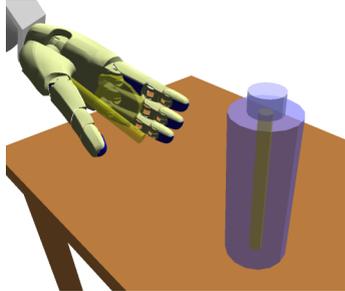
Each phase is started when all the constraints of the preceding phase are fulfilled. Some constraints should be fulfilled throughout a movement phase, others only at the end of it. For the latter, the robot has the freedom to choose a smooth trajectory.

Several constraint directions were reused in these phases, some with the same desired values, others with different values. But I also changed the orientation representation of the spatula: During the pushing-phase aligning front- and side edge with the oven is the most concise expression. During lifting and turning, the aligning the blade normal with gravity is easier and more flexible.

3.4.0.2 Grasping a bottle

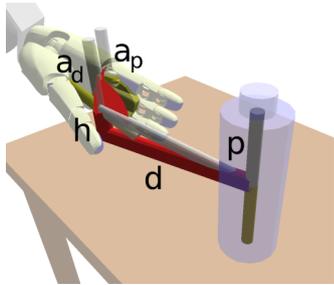
In this example, the robot hand is annotated with features and a round bottle is grasped. The symmetry axis of the bottle is exploited as well it's height.

The following features defined:



hand:	bottle:
palm	axis
grasping-direction	

Using these features, the following constraints are defined:



- (p) $\text{pointing-at}(\text{hand}_{\text{direction}}, \text{bottle}_{\text{axis}}) = 0$
- (a_p) $\text{perpendicular}(\text{hand}_{\text{palm-normal}}, \text{bottle}_{\text{axis}}) = 0$
- (a_d) $\text{perpendicular}(\text{hand}_{\text{direction}}, \text{bottle}_{\text{axis}}) = 0$
- (h) $\text{height}(\text{hand}_{\text{direction}}, \text{bottle}_{\text{axis}}) \in [-r_{\text{bottle-axis}} \dots r_{\text{bottle-axis}}]$
- (d) $\text{distance}(\text{hand}_{\text{direction}}, \text{bottle}_{\text{axis}}) = 0$

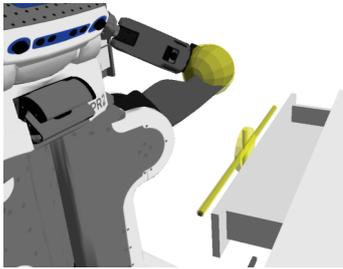
The constraints (p), (a_p) and (a_d) avoid crashes with the fingers: The constraints (p) makes the hand 'point' at the spatula, (a_p) holds the palm vertical and (a_d) holds the fingers horizontal. The distance (d) is first set to $r_{\text{bottle}} + r_{\text{fingers}}$ as an approach pose, then the hand closes in to zero.

3.4.0.3 Closing a Drawer

In this the PR2 uses it's elbow to close a drawer in the kitchen. The elbow is under-actuated in that only the position of the elbow can be moved on a sphere, but for closing the drawer, the 3D position of the elbow is of interest. However, since the contact point on the drawer can be anywhere on it's front board, there is enough freedom in the task to perform this movement.

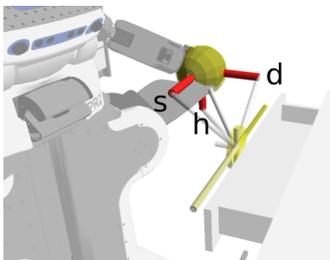
3. Generation of movement specifications

The chosen features for this task were the elbow as a point, the front surface of the drawer and its lateral and vertical dimensions:



arm:	drawer:
elbow	front-plane
	drawer-width
	drawer-height

I span only three constraints representing Cartesian coordinates of the elbow with respect to the drawer: The side-distance (*s*) and vertical distance (*h*) are set to the drawer dimensions. The pushing direction (*d*) is controlled to 0 in order to close the drawer:



- (s) $\text{height}(\text{elbow}, \text{drawer}_{\text{width}}) \in [-r_{\text{drawer-width}} \dots r_{\text{drawer-width}}]$
- (h) $\text{height}(\text{elbow}, \text{drawer}_{\text{height}}) \in [-r_{\text{drawer-height}} \dots r_{\text{drawer-height}}]$
- (d) $\text{height}(\text{elbow}, \text{drawer}_{\text{front}}) = 0$

The feature-constraint based task description captures all task freedoms, which is necessary in this case to perform the drawer-closing task successfully.

Execution of movements

After discussing various ways to generate movement- and constraint specifications, the next sections focus on the more technical aspects of the movement- and constraint representations. They compare two approaches to combine constraints into an executable form, present the controller framework and various secondary tasks which are executed in the null space of the movement. I present and evaluate methods for conflict and singularity detection and evaluate how our constraint-based movement description extends the robot's work space significantly. Finally, an implementation is proposed for translating force constraints into a task-specific stiffness matrix and -frame. The effectiveness of modulating the robot's impedance in this way is demonstrated experimentally.

4.1 Task Functions

After a constraint-based movement description has been generated, we now study the implementation of the task functions, which were defined rather abstractly in Section 3.2. They are the core of the movement execution framework but are also needed to perform conflict checking and monitoring.

The goal of these functions is to define meaningful movement directions in which we can specify movement goals. For example, a task function might express, how well the spatula is pointing towards the pancake. The corresponding (instantaneous) movement command would be to 'point more/less towards the pancake'. In this sense, task functions are a coordinate transformation that abstracts away the robot and environment. Preferably, these relations are chosen such that they can be used in several steps of a movement, but this is not necessary.

A task function must depend on an object's pose or the robot joint angles (otherwise, it can't be controlled). It can be vector-valued and express several relations.

The aforementioned instantaneous movement directions are computed as the derivative of the task function, which is why a task function must be differentiable. This means, that binary true/false expressions cannot be used as task functions, these must be converted to a measure that expresses how far the current situation is from fulfilling the expression.

For instance, the pointing relation from above could be implemented as the angle between the tool's main direction and the line connecting the tool and object. When this angle is zero, then the tool is pointing towards the object.

In this chapter, I will analyze two approaches for expressing task functions: Virtual Kinematic Chains are used as an instance of the Feature Frame Approach, a modeling approach that is used in the iTaSC framework. I will describe the specification process with respect to possibilities to automate the selection, placement, and perceptual and semantic grounding. The second approach, based on elementary 'feature functions', is a more direct alternative which directly relates two features to each other. Then, I describe how to visualize task functions, how to test sets of constraints for conflicts and singularities and conclude with a discussion on controlling alignment using task functions.

4.1.1 Virtual Kinematic Chains

One possibility to define task functions is used in the iTaSC framework (Smits et al., 2008; De Schutter et al., 2007): The relative pose between object and tool is spanned by a Virtual Kinematic Chain (VKC), consisting only of prismatic and revolute joints and no fixed offsets. This chain models the six degrees of freedom of the rigid transformation between object and tool. The chain is chosen such that its joints resemble intuitive constraints. Some of these virtual joints are used for control, while the others are left free for different constraints or optimization.

This representation is an instance of the Feature Frame Approach (Mason, 1981; De Schutter and Van Brussel, 1988a; Bruyninckx and De Schutter, 1996), which is mature and well tested in the iTaSC framework. It has the advantage, that both, freedoms and constraints are explicitly represented in a geometrically consistent form, in which all six DOFs between two objects are accounted for. The Feature Frame Approach can express constraints over up to four moving entities (two objects and two features), enabling it to model e.g. laser pointing tasks (De Schutter et al., 2007). This approach also has a formal way to model uncertainties, and correct them using on-line sensor readings. However, this feature currently requires the manual specification of an appropriate measurement model.

A virtual kinematic chain that exactly covers all six degrees of freedom of rigid transformations, obviously has exactly six axes. I consider as possible joints translations along the x- y- and z-axis and rotations around the same axes. Using these six joint types, there are $6^6 = 46656$ possible VKCs. By checking, whether the Jacobian matrix of these chains are non-singular – so that they cover the 6-D space of translation and rotation – 5664 chains remain.

But of these chains, not all are unique: For example, a rotation around the x-axis and a translation along the x-axis can be done in any order. Thus, when permuting the respective joint angles, these chains are kinematically equivalent. This can be found by evaluating the forward kinematics $f_{chain}(\mathbf{q})$ and comparing the results for a number of random joint angles. The first chain is evaluated with random joint angles $\mathbf{p}_1 = f_{chain_1}(\mathbf{q}_r)$. The second chain is evaluated with all K permutations of the same angles $\mathbf{p}_2^k = f_{chain_2}(perm_k(\mathbf{q}_r)), k = (1..K)$. If a permutation k is found for which the forward kinematics computes the same pose, i.e. $\mathbf{p}_1 = \mathbf{p}_2^k$, then the two chains

4. Execution of movements

are considered equivalent. This result is checked with several sets of random joint angles. After this test, 2927 unique virtual chains are left.

The full-rank virtual kinematic chains have at least one and at most three prismatic joints (Figure 4.1). The remaining 3-5 joints are revolute.

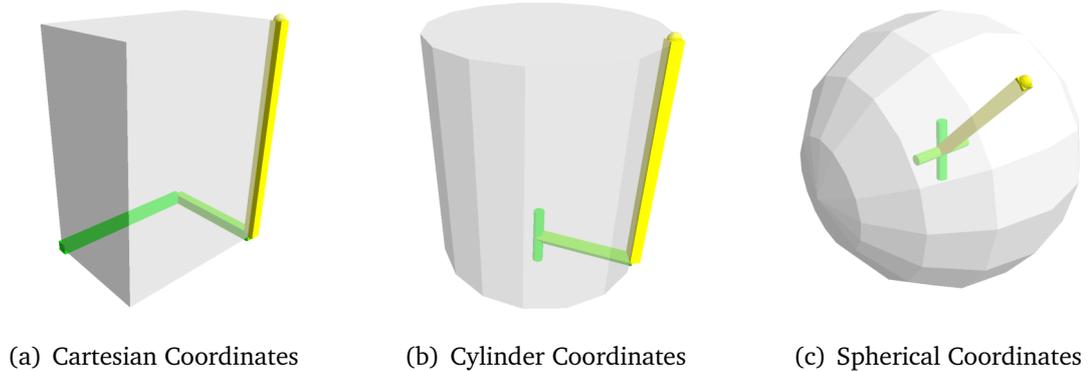


Figure 4.1.: *Coordinate Systems*

A representative with one prismatic joint are Spherical Coordinates, two prismatic joints are used in Cylinder Coordinates and three prismatic joints are the x- y- and z-translation of Cartesian Coordinates. Each of these chains needs to be combined with some kind of Euler angle rotations to account for the orientation (Diebel, 2006).

The above consideration suggests, that many of the VKCs can be understood as a composition of a positioning part and an orientation part. However, among the 2927 unique chains, only 312 can be separated into a positioning part (i.e. three joints that cover 3d positioning) and an orientation part (i.e. three joints that cover orientations). This property was tested as follows: The chain is split into the first half, containing the first three joints of the chain, and the second half, containing the last three joints. Whether such a three-axis sub-chain 'covers 3d positioning' is checked by computing the rank of the first three rows \mathbf{J}_t of its Jacobian \mathbf{J} . Whether a sub-chain 'covers orientations' is checked by computing the rank of the last three rows \mathbf{J}_r of its Jacobian \mathbf{J} :

$$\text{'covers positioning'} \Leftrightarrow \text{rank}(\mathbf{J}_t) = 3$$

$$\text{'covers orientation'} \Leftrightarrow \text{rank}(\mathbf{J}_r) = 3$$

$$\text{with } \mathbf{J} = \begin{bmatrix} \mathbf{J}_t \\ \mathbf{J}_r \end{bmatrix}.$$

Table 4.1 summarizes the results of the previously described analysis. It can be seen

# translational joints	1	2	3	total
full rank chains	288	2208	3168	5664
unique chains	240	1296	1391	2927
separable chains	144	216	144	504
unique separable chains	144	144	24	312

Table 4.1.: *Number of different Virtual Kinematic Chains*

that still many possible virtual kinematic chains exist. It is the planners task to choose the right chain for a given task, using hints like object symmetries.

In my approach, I pick a positioning sub-chain for the object and an orientation sub-chain for the tool.

For the object, I select the sub-chain by the symmetry: I choose Cartesian coordinates when the object is of cuboid shape, cylinder coordinates for objects with an axis of symmetry and spherical coordinates for spherical objects.

The tool is assigned RPY coordinates, so that the rotation around up to three perpendicular tool edges can be controlled. If the tool exhibits symmetry (like the skimmer in Figure 3.3), this axis can be set to 'free'. Note, that this specification assumes a relatively small tool since only a single tool position can be expressed using the positioning sub-chain.

With effectively only three choices, which are taken based on object symmetries, the selection of the kinematic chain is relatively simple. But it also leaves a big gap between the number of possible virtual kinematic chains and what can be inferred semantically. In the example of pancake-flipping, there is a rotational symmetry, both in the pancake and the oven. I therefore select cylinder coordinates for positioning.

A more interesting task is the aligning and positioning of the coordinate frame on the object and tool.

Let us first investigate, how a coordinate frame can be aligned to an object: It should be oriented such that task-relevant surface normals and edges are aligned with coordinate axes. The same kind of alignment is required in the Task Frame approach which was mentioned earlier. Take the example of the spatula in Figure 4.2: It has a

4. Execution of movements

	d_b	d_s	d_l	d_r	d_f
d_b	I	\perp	\perp	\perp	\perp
d_s		I	\parallel	\parallel	\perp
d_l			I	\parallel	\perp
d_r				I	\perp
d_f					I

Table 4.2.: *The parallel / perpendicular relations between feature directions.*

front edge with direction d_f , two side edges with directions d_r and d_l , a blade with a surface normal d_b and a symmetry axis with direction d_s . Of these directions, up to three can be chosen to align a coordinate frame.

However, some of these geometric features are parallel and are thus redundant for the alignment process.

Table 4.2 shows the parallel/perpendicular relation between the feature directions. These relations induce the graph in Figure 4.2, where nodes are the feature directions and the edges are the parallel / perpendicular relations.

The frame axes are all perpendicular to each other, their relations are depicted in Figure 4.3. Aligning a coordinate frame with a tool now amounts to finding this frame graph as a sub-graph in the feature graph.

Since the directions d_s , d_l and d_r are parallel, they can be combined into one node, as in Figure 4.4. After this simplification, it becomes clear to which directions the

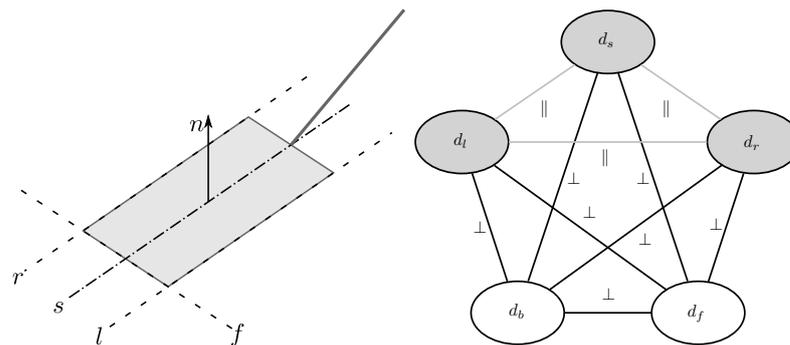


Figure 4.2.: *Geometric features of a spatula and their parallel / perpendicular relation graph*

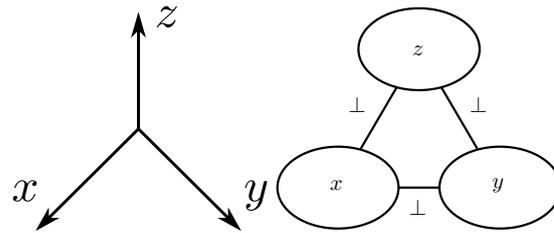


Figure 4.3.: Geometric feature relations of an abstract coordinate system.

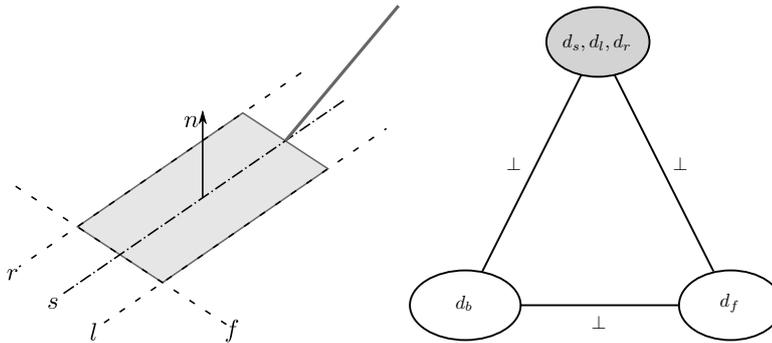


Figure 4.4.: Reduced geometric feature relation graph

coordinate system must be aligned. But the concrete assignment of the x-, y- and z-axis is still open.

What can be done, if such a sub-graph does not exist, such as in Figure 4.5? In this case, a coordinate frame can be completed by computing the cross product of the other two axis directions. In this case, only two perpendicular directions are needed.

If the directions are neither perpendicular nor parallel, then no connection is made in the graph. This can lead to disconnected graphs, but the sub-graph matching can still be applied to the sub-graphs. In this case, the angle criterion for 'perpendicular' can

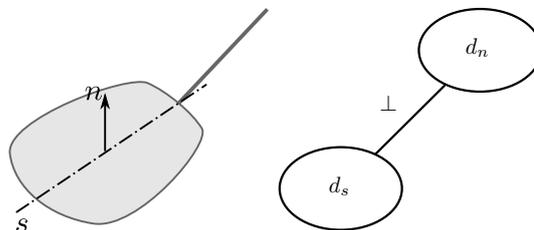


Figure 4.5.: Rounded spatula with only two geometric features: A symmetry axis and a blade normal vector.

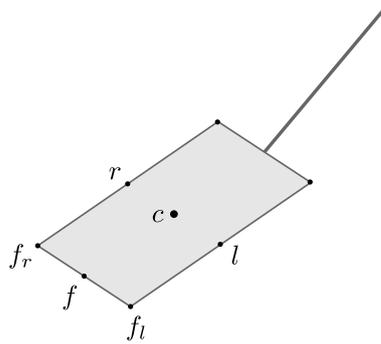


Figure 4.6.: *Points on a spatula that are relevant for frame positioning*

be relaxed, until a sub-graph for a coordinate frame exists. If several matches exist, then one must be chosen according to the priority of the involved features.

Let us now turn our attention to the placement of the frame's origin \mathbf{o} . Again, I consider as input the geometrical features of the tool: The mid-points of the edges r , l , f and the corners f_r and f_l , which are also the intersections of the edges f with r and f with l . The point c is the center point of the blade and can be seen as the 'location' of the blade's plane segment feature.

When expressing a constraint that involves the location of a feature, then placing the origin on it is a natural choice. Therefore, the corners f_r or f_l appear to be ideal, since they lie on two edges and could thus express constraints involving both features. However, it might be needed to place the origin between two or more feature points. In a more general form, I could express the coordinate origin in barycentric coordinates of the relevant features, i.e. as a weighted sum of the feature locations:

$$\mathbf{o} = \alpha c + \beta f_r + \gamma f_l + \dots, \quad \alpha + \beta + \gamma + \dots = 1$$

The choice of the weights $\alpha, \beta, \gamma, \dots$ determines the location of the origin: This form allows to select a single feature point, by setting its coefficient to 1 and all other coefficients to zero. But they can also express the center between any number of selected features.

However, force-controlled alignment constraints have different requirements. They are implemented as zero-torque constraints perpendicular to the feature. In order to make these constraints work, the origin needs a lever to the potential contact

points. So these corners must be avoided if such an alignment strategy is used. In fact, the origin's distance to these corners and edges should be maximized for optimal alignment results (See section 4.3.6.1).

The final step in the chain specification is to label the chains axes in order to make the connection back to the planner. This labeling is based on the chain type, the placement of the end frames and the involved features. For example, if a cylinder-coordinate chain is aligned such that the cylinder axis is vertical, then the semantic annotation 'height' can be made to the respective translational joint.

For the pancake-flipping task, I placed the 'object' frame on the oven center, with the z-axis pointing up. The 'tool' frame was placed at the center of the spatula, with the z-axis pointing forward and the x-axis pointing up, i.e. in the direction of the blade normal. This allows to express alignment tasks as zero-torque constraints, since all corners and edge have a 'lever' to the frame center point. I then used cylinder coordinates to express the position of the spatula with respect to the oven (because of the symmetry of the pancake oven), and defined RPY-angles to express the orientation of the spatula with respect to the oven. In order to find the right linkage joint when defining constraints, I semantically annotate the joints of the virtual kinematic chain. For the pancake-flipping task, the first three axes of the linkage, which form cylinder coordinates, are labeled "angle", "distance" and "height", while the last three axes are labeled "roll", "pitch" and "yaw". Especially the orientation axes must be linked back to the alignment of the tools features, e.g. setting "roll" to zero means aligning the front edge of the spatula with the oven plane. This knowledge can be encoded in the elementary constraints.

In order to use these virtual kinematic chains in a task-function based controller, a position-inverse kinematics is required in order to compute the current value of the task function. Using numerical, this is not real time safe, but has been proven to be fast enough, once a first solution was found. An analytical solution is real time safe but requires manual effort. However, given that I currently only select from three possible chains, the effort is limited.

Summing up, the Virtual Kinematic Chains allow for a versatile task representation which allows to control single, task-related degrees of freedom. All six dofs are explicitly represented in the chain, but some may be inactive, leaving some freedom

4. Execution of movements

for the execution. Having the free dofs represented explicitly is an advantage for the high level reasoning which can consider them for constructing more constraints.

On the other hand, this approach is very indirect and requires heuristics for the selection, placement and alignment of the virtual kinematic chain. Furthermore, from the large number of VKC's I could pick only three choices and the alignment of the end frames has to take into account the constraints that will be defined on the chain.

4.1.2 Feature-Functions

Instead of selecting and placing a virtual kinematic chain that must fit several constraints, I propose to construct the task function out of *feature functions* which map two geometric features onto a scalar value. I then stack these feature functions to form the task function. An example of such a feature function is 'aligned(spatula_{front-edge}, oven_{plane})', where 'aligned' is the feature function, 'spatula_{front-edge}' is a feature on the tool and 'oven_{plane}' is a feature on an object. These functions carry the semantic annotation in their names like 'perpendicular' or 'pointing-at'.

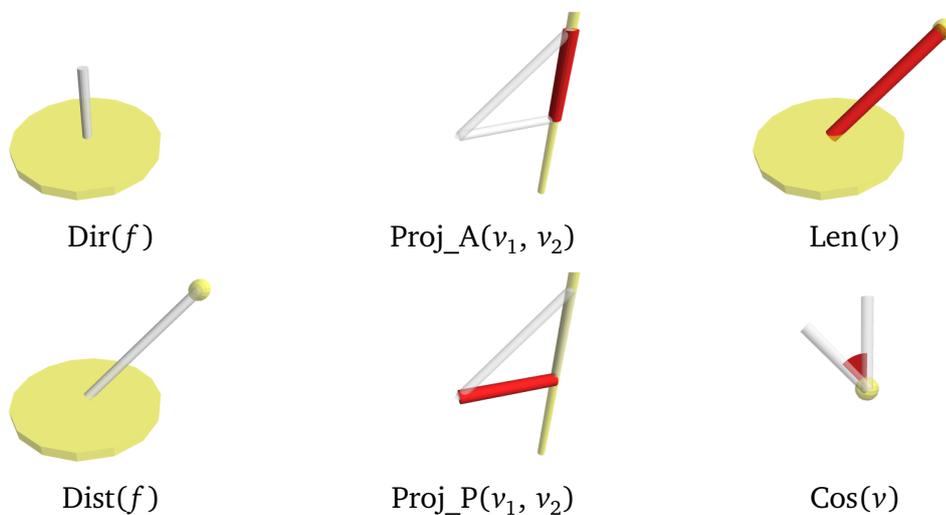


Figure 4.7.: *The geometric functions used to construct feature functions.*

This decomposes the task function into the same units that are reasoned about on the planning level and the specification process is simplified down to the selection of two features and a feature function. It is a much more direct approach, compared to selecting, placing and annotating virtual kinematic chains. By defining the

feature functions on the geometric features themselves, I achieve a direct grounding in perception. By construction, no unnecessary dependencies between constraints are introduced and some singularities are avoided. This is shown later in Section 4.1.7.

A feature function usually depends on the pose of the robot because at least the feature of the tool is movable by the robot's actuators. Moreover, the feature function must be differentiable with respect to the movements of the robot. As a result, the calculated gradient can be used to build the interaction matrix \mathbf{H} , which in turn is used to control the robot such that it causes the feature function to converge to a desired value y_d . Feature functions can be implemented in arbitrary ways, as long as they fulfill the requirements mentioned above. They have the signature (Feature \times Feature $\rightarrow \mathbb{R}$). In order to represent the geometric reasoning that leads to a particular feature function, I express them by composing functions of three different signatures:

- (1) Functions that convert geometric features into vectors (Feature \rightarrow Vector),
- (2) Functions that operate on vectors (Vector \times Vector \rightarrow Vector) and
- (3) Functions that convert vectors into scalars (Vector \times Vector \rightarrow Scalar), which are returned by the feature function.

The first group of functions accesses the features' vector data, the second group performs some basic geometric reasoning while the third group computes the result.

I use the following functions:

- Group (1):
 - $\text{Dir}(f)$: Extract the direction of a line feature or the normal direction of a plane feature.
 - $\text{Dist}(f_1, f_2)$: Compute the distance vector between the feature centers.
- Group (2):
 - $\text{Proj_A}(v_1, v_2)$: Project the vector v_1 onto vector v_2 (*align*).
 - $\text{Proj_P}(v_1, v_2)$: Project the vector v_1 *perpendicular* to vector v_2 .
- Group (3):

4. Execution of movements

- $\text{Len}(v)$: Compute the vector length of the vector v .
- $\text{Cos}(v_1, v_2)$: Compute the cosine of the angle between the vectors v_1 and v_2 .

This approach eases the readability of the functions and makes it transparent whether orientation, position or both from the object and tool are used. However, this property is also reflected in the interaction matrix and can be computed numerically for any task function.

My system contains – among others – the following feature functions which will be used for evaluation. In the following list, f_t denotes a feature on the *tool* and f_w denotes a feature on an object in the *world*:

- $\text{perpendicular}(f_t, f_w) = \text{Cos}(\text{Dir}(f_t), \text{Dir}(f_w))$, equals zero if both feature directions are perpendicular to one another
- $\text{height}(f_t, f_w) = \text{Len}(\text{Proj_A}(\text{Dist}(f_t, f_w)), \text{Dir}(f_w))$: corresponds to the length of vector between the positions of the two features, projected onto the direction of the world feature
- $\text{distance}(f_t, f_w) = \text{Len}(\text{Proj_P}(\text{Dist}(f_t, f_w)), \text{Dir}(f_w))$, denotes the length of the vector between the positions of the two features, projected perpendicular to the direction of the world feature
- $\text{pointing_at}(f_t, f_w) = \text{Cos}(\text{Proj_P}(\text{Dist}(f_t, f_w), \text{Dir}(f_w)), \text{Proj_P}(\text{Dir}(f_t), \text{Dir}(f_w)))$, equals zero if the main direction of the first feature is pointing at the second one, i.e. is directed at some point on the line defined by the position and direction vectors of the second feature.

Whenever possible, I specify the feature functions in a minimal way and base it on as few features as possible. This way the specification process remains simple and I avoid unwanted dependencies between constraints.

Consider the implementation of the *perpendicular* function in Figure 4.8. It becomes 0 when the vectors involved are perpendicular, 1 when they are aligned and -1 when they are opposite. This can be computed using the dot product, which is well defined in 3D. However, when the vectors are aligned (or in opposition) it is impossible to

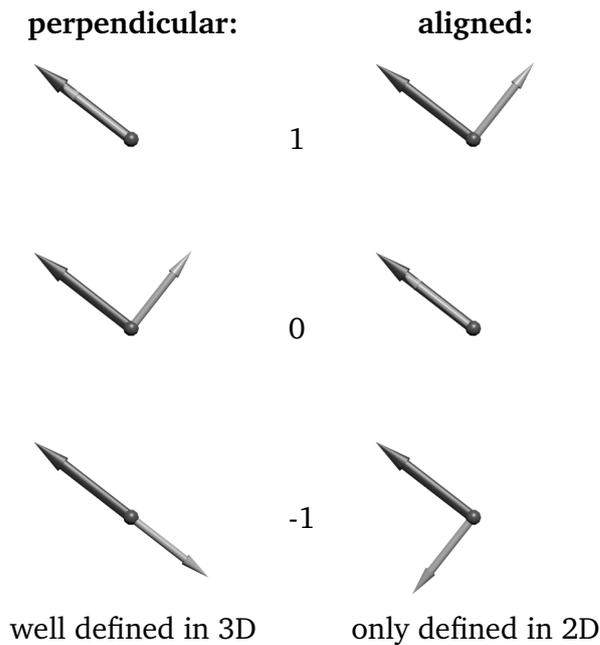


Figure 4.8.: Comparison of 'perpendicular' and 'aligned' feature functions

specify in which direction one vector should 'turn away', any movement command is ambiguous. However, when limited by other constraints, a predictable movement is still possible. For this purpose, a more convenient function would be *aligned*, where the aligned position yields 0, turning left yields negative values and turning right yields positive values. However, 'turning left' and 'turning right' is only defined in a plane, with respect to an 'up'-reference). This direction is required as an extra input. The same applies to task functions that yield an angle that ranges from 0° to 360° : When an angle α is to be distinguished from its counterpart $360^\circ - \alpha$, then this distinction depends on the point-of-view, which is used in the computation either explicitly or implicitly.

Compared to Virtual Kinematic Chains, a Feature Function-based constraint is much easier to specify (see Figure 4.9). For a VKC I must choose a chain with exactly six axes, construct coordinate frames and then assign semantics to each axis. A Feature Function refers directly to geometric features which are grounded in the perception, and the functions carry their abstract semantics in their name. Moreover, the number of Feature Functions that can be specified is not limited.

4. Execution of movements

```
start_frame = pancake
end_frame = spatula
VKC = {rot-z,trans-x,trans-z,
       rot-x,rot-y,rot-z}
constraint1{
  semantics = "approach-angle"
  range <unconstrained>}
constraint2{
  semantics = "distance"
  range = [...]}
constraint3{
  semantics = "height"
  range = [...]}

...

constraint6{
  semantics = "align-front"
  range = [...]}

constraint{
  tool_feature = spatula-blade-plane
  object_feature = pancake-plane
  function = perpendicular
  range [...]}
constraint{
  tool_feature = spatula-axis
  object_feature = pancake-center
  function = pointing-at
  range [...]}

...

constraint{
  tool_feature = spatula-front-axis
  object_feature = pancake-rim-line
  function = distance
  range [...]}
```

Figure 4.9.: Comparing VKC based (left) and Feature Function based movement descriptions (right). Whereas the virtual kinematic chains must be annotated to connect them to 'high-level' concepts like a tool-object distance, the Feature-Function based movement description carries this information by construction.

On the other hand, this flexibility also allows the construction of conflicting constraint sets. In order to detect and eliminate such conflicts, there is a conflict check which I describe in Section 4.1.4.

4.1.3 Visualization of Task Functions

The last section has specified very simple elementary task functions that can express relations like distance or alignment. Then, several of these feature functions were composed to form useful movement specifications. The next step is to examine, how these functions behave in concert when they are controlled towards desired values. For example, aligning the front edge of the spatula to the oven plane can be achieved by many different rotations: The feature function specifies a direction of the rotation axis, in which the angle changes fastest. However, that axis could be placed on the center of the spatula, its front edge, the oven center or any other point in the world. But depending on the placement of that axis, this constraint would influence other constraints differently. Assuming some kind of 'independence' of the constraints, the controller instantaneously finds a movement (a twist) for each feature function that

does not affect the others. I analyze the result of this optimization and compile this analysis into a visualization which serves as a 'mind's eye' on the movement execution: It displays the twist that is caused by changing a constraint. The visualization for this interaction works for *any* task function that is defined over Cartesian transformations, not only virtual kinematic chains whose looks they resemble. My method can even be applied to a robot Jacobian (Figure 4.12): In this case, the axes of the actuators are displayed, which allows for a quick visual verification that all computations and coordinate transformations are correct.

The visualization gives insight into the movement execution: Properties like alignment or changing locations of movement axes can not only be visually observed, but also enable monitoring of the soundness and effectiveness of the execution of such a constraint set.

The input for my method is merely the task function interface, consisting of the interaction matrix \mathbf{H} and the poses (more specifically, the positions \mathbf{p}) that the task function relates to each other. This is the reason why this method applies to any task function defined over Cartesian transformations.

In the following paragraphs, I first consider how to compute appropriate, independent twists out of \mathbf{H} . Then I explain how to visualize twists by their instantaneous rotation axis. Finally, I cover how to display pure translational twists as line segments, by extracting their directions and appending them so that they connect to a chain that spans the translation \mathbf{p} from object to tool.

4.1.3.1 Computing Independent Twists

For my visualization method, I first must compute instantaneous movements, or twists, that affect one and only one constraint at a time. By this definition, I call these twists *orthogonal*.

They are more useful for display purposes than the rows of the interaction matrix \mathbf{H} . In particular, the angular alignment constraints do not depend on a position and thus the location of their instantaneous rotation axis is not defined. However, by demanding orthogonality with the other constraints, the location of the axis can be

4. Execution of movements

inferred: It must be placed where a rotation around it does not affect any other constraint.

The framework computes the task interaction matrix for a set of constraints, which describes how a particular twist affects these constraints. Here, I seek the inverse: Given a particular task space velocity $\dot{\mathbf{y}}$ (one, where only one constraint is non-null), I want to compute the twist that affects this (and only this) constraint. This is computed by the inverse of the interaction matrix \mathbf{H}^+ .

If the twists in the rows of \mathbf{H} are *linearly independent*, the inverse matrix \mathbf{H}^+ yields columns of mutually *orthogonal* twists where each twist affects only its respective constraint.

This matrix contains the desired twists. In the control equation this inversion happens implicitly when the control law is solved for the joint velocities (see Section A.1).

4.1.3.2 Visualization of Twists

For visualizing twists, I draw some ideas from screw theory (Mason, 2001) and Plücker line representations (Shoemake, 1997). A twist can be interpreted as a rotation around an axis in space and a simultaneous translation along the same axis (a screw movement). For rotational joints, this axis can be visualized (see Figure 4.10): The segment of the line that is closest to the origin is displayed using a cylinder.

For pure translational movements, the location of this axis is not defined, only its orientation. This type of movement can be visualized as a line, showing the direction. However, it still must be decided where to place this line and how long it should be.

When displaying twists, three possibly different coordinate systems are of interest:

- The reference frame, in which directions are expressed
- The reference point, around which pure rotations happen
- The target frame, which governs what piece of the line is shown

The reference frame and -point must be known in order to interpret a twist. The target frame is just necessary for visualization purposes. The formulas for changing the

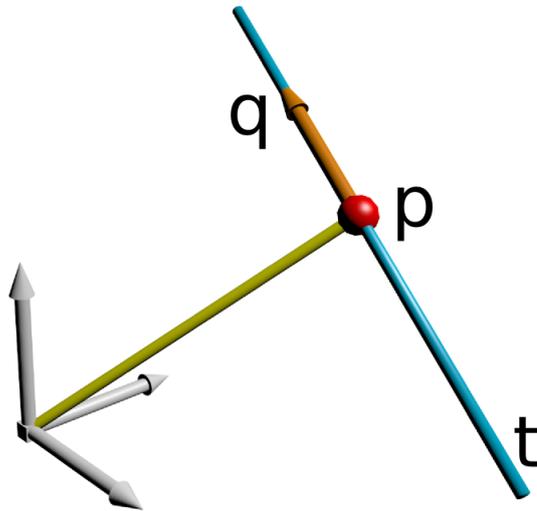


Figure 4.10.: *Visualization of a twist as the instantaneous axis of rotation (q is the direction, p is the closest point to the origin)*

reference frame and $-$ point are described in Section A.1. Using these transformations, the twist is transformed into the target frame.

Visualization of Rotational Twists. After correctly transforming the twists of the inverted interaction matrix \mathbf{H}^+ , I now consider how to visualize a rotational twist by displaying its instantaneous rotation axis.

For this visualization, I exploit the fact that a twist can be re-interpreted as a Plücker line $\mathbf{t} = (\mathbf{q}, \mathbf{q}_0)$ where \mathbf{q} is the rotational part and \mathbf{q}_0 is the translational part of the twist (Mason, 2001). The direction of the line is simply its directional part \mathbf{q} and the position is the closest point of the line to the origin and is computed as:

$$\mathbf{p} = \frac{\mathbf{q} \times \mathbf{q}_0}{\|\mathbf{q}\|^2} \quad (4.1)$$

A short line segment around this point is displayed to visualize the twist.

Visualization of Translational Twists. If the twist is (nearly) a pure translation, then \mathbf{q}_0 is close to 0 and the axis would be placed (nearly) at infinity. For these constraints, only the direction vectors $(\mathbf{v}_1, \dots, \mathbf{v}_n)$ can be used, locations and lengths need to be inferred.

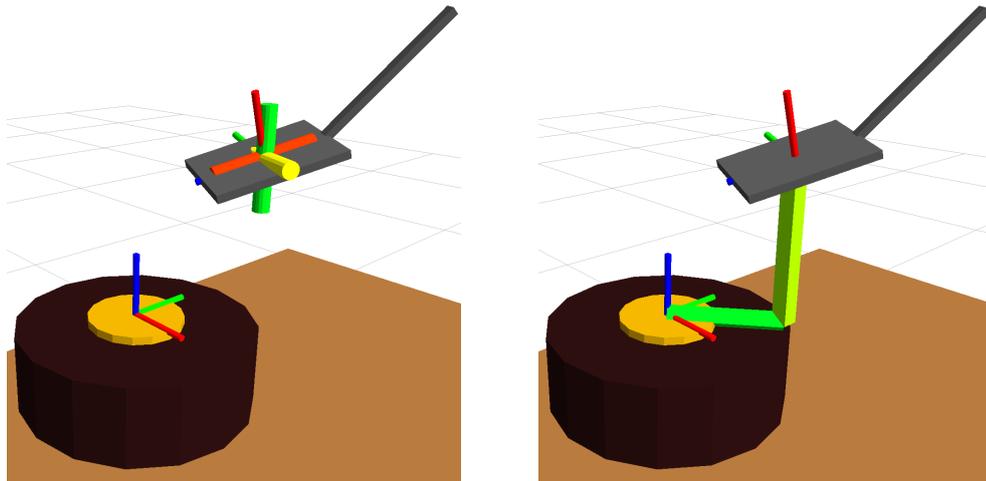


Figure 4.11.: Visualization of rotational (left) and translational (right) axes of a task function. The axes are color coded from green to red, with red representing high task velocities and green representing low task velocities, i.e. a constraint that is fulfilled.

This step takes into account the translation \mathbf{p} between tool and object: I require the translations to form a path from the object origin to the tool origin, i.e. they must sum up to \mathbf{p} :

$$\mathbf{p} = \sum_{i=1}^n \alpha_i \mathbf{v}_i \quad (4.2)$$

In matrix form this becomes:

$$\mathbf{V}\alpha = \mathbf{p} \quad (4.3)$$

with $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_n]$. Solving this linear equation for α yields the lengths of the visualization vectors. Their locations are determined by concatenating these vector together in the order as they appear in the inverse interaction matrix \mathbf{H}^+ (see Figure 4.11, right).

I also move the target frame of the rotational constraints to the end point of the last translational constraint line, in order to maintain visual coherence and to avoid axis segments far away from other axes.

This visualization is especially well suited for virtual kinematic chains and other task functions that fully cover the 6 degrees of freedom between object and tool.

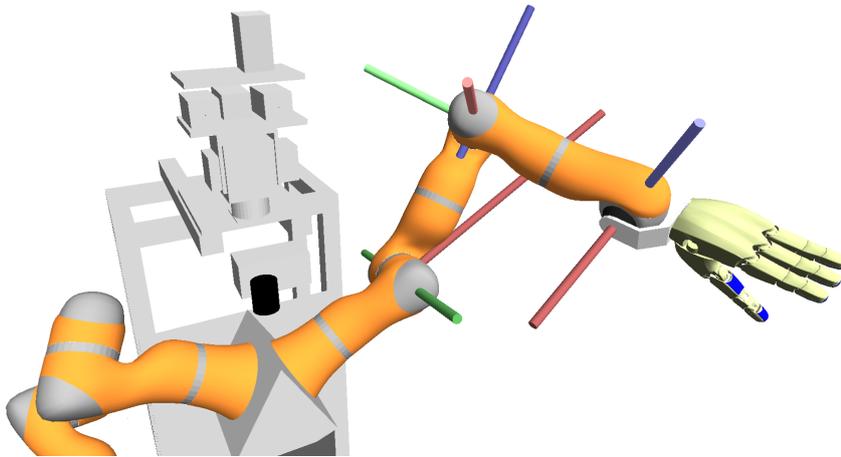


Figure 4.12.: *Robot joint axes, extracted from Jacobian matrix*

With the given interface – the interaction matrix \mathbf{H} and the translation \mathbf{p} between object and tool – this approach applies to all task functions which are defined over 3D poses. Even unmodified, my method correctly displays the rotation axes of a robot arm, although some of these axes might be floating in the air (Figure 4.12).

4.1.4 Dependency Evaluation

Having defined feature functions and their graphical evaluation, I now focus on checking the soundness of a combination of constraints, which I posed as a requirement in Section 2.3.5.

Imagine a situation in which a user of my system has assigned alignment constraints to both the left and right side of the spatula. Are there ways to detect that they depend on each other, as shown in Figure 4.13 (a)?

Another example are the alignments of the spatula’s front edge, its side edge and its blade (all with respect to the oven). Only two of them are controllable at the same time (see Figure 4.13 (b)).

This kind of reasoning is crucial when combining constraints in a modular fashion as I propose to allow choosing independent constraints. This means that each constraint can be controlled independently. Such a situation is given when the rows of the inter-

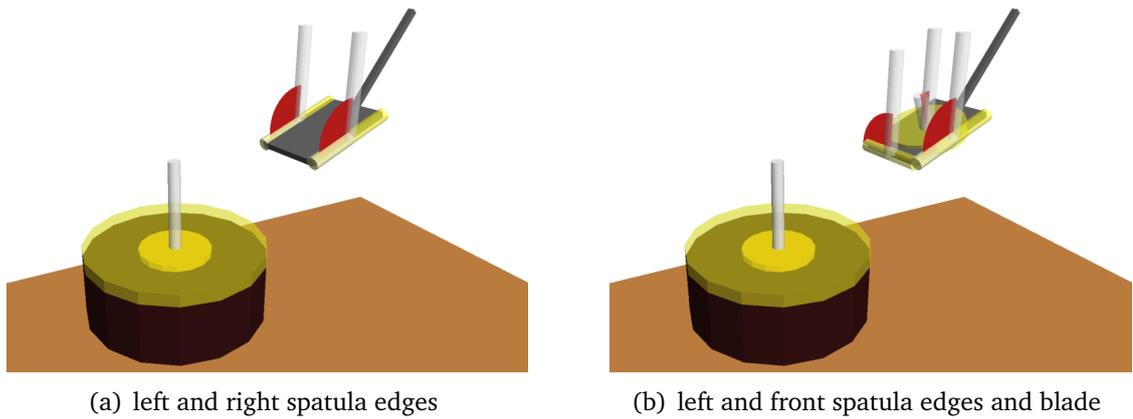


Figure 4.13.: *Dependent alignment constraints*

action matrix \mathbf{H} are linearly independent. In this case it is possible to find an inverse matrix \mathbf{H}^+ in which each column contains a twist that affects one and only one constraint. In order to rule out local singularities – which might be avoided anyway by the desired ranges of the constraints – the algorithm checks the rank of \mathbf{H} for many transformations and take the maximum.

Another interesting question that can be answered with this tool is whether two sets of constraints are equivalent from a control point of view, i.e. if they are controlling the same degrees of freedom. This is true if

$$\text{rank}(\mathbf{H}_1) = \text{rank}(\mathbf{H}_2) = \text{rank}\left(\begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{bmatrix}\right) \quad (4.4)$$

By using only the interaction matrix, these methods can be used with any task function. In particular, they can be used to compare Virtual Kinematic Chains with sets of Feature Functions: If the rank reveals, that some rows of a VKC interaction matrix are linearly dependent to some other rows of a Feature Function interaction matrix, then the respective joints of the VKC express the same movements as the respective Feature Functions. This check is implemented in the shared library which also implements the feature functions themselves. In addition, a ROS service is available which checks a constraint set for conflicts.

4.1.4.1 Finding minimal constraint sets

If, for some reason one is confronted with a redundant set of constraints, then a straight-forward approach can be used to remove unnecessary constraints:

1. Remove one constraint at a time from the constraint set
2. Recompute the Jacobian rank, for several tool-object poses.
3. If the (maximum) matrix rank stays the same, then the removed constraint was redundant. If several such constraints can be found, then the 'least important' constraint is removed.
4. Continue at step 1 until the number of constraints equals the Jacobian rank.

This algorithm requires a notion of importance for a constraint. Such a ranking may be based on the features that it refers to, or the feature function. For constraints that are derived from movement data, the fitting error might be taken into account.

In Figure 4.14, this elimination process is illustrated: In addition to the previously described elementary task functions for pancake flipping, there are three extra functions: The distance of the right corner of the spatula to the center of the oven (`dist_right_corner`), the distance of the left corner of the spatula to the center of the oven (`dist_left_corner`) and the angle of the spatula blade with the oven plane (`align_blade`). The two additional distance functions are redundant with the distance of the spatula center to the oven center. The difference of those two distances is redundant with the `pointing_at` function. And the blade alignment function is redundant with the front edge- and side-edge alignment functions. In the middle row, the three additional task functions are removed: Each removal keeps the rank of the combined Jacobian at six. In the top row, the functions `dist`, `pointing_at` and `align_side` are removed – their degrees of freedom are encoded in the last three functions: `distance_left_corner` and `distance_right_corner` encode the distance and `pointing_at`. The functions `align_blade` and `align_front` encode the same degrees of freedom as `align_front` and `align_side`. The last row shows, how the `height` function will not be touched by the algorithm, since its removal would decrease the feature Jacobian rank to 5. This function is non redundant and must remain in the constraint set.

4. Execution of movements

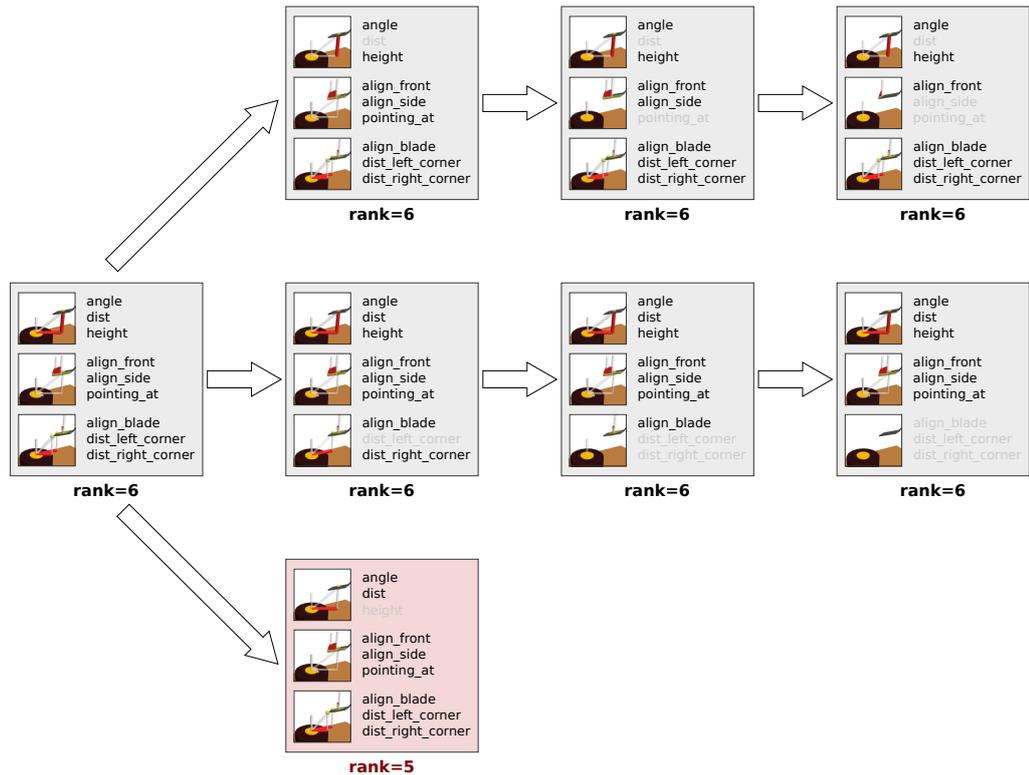


Figure 4.14.: *Sample constraint elimination processes: Top row: The distance-, pointing_at- and align_side task functions are removed. Part of the spatula orientation is described by the distance of the spatula corners to the center. Middle row: The extra task functions distance_left_corner, distance_right_corner and align_blade are removed. This yields the VKC-like task function as described previously. Bottom row: The height task function cannot be removed without lowering the rank. This task function is non-redundant.*

4.1.4.2 Completing constraint sets

There is one other advantage of virtual kinematic chains over feature functions that was not discussed yet: VKCs always have an explicit notion of the free degrees of freedom that are not actuated by the current constraint set, but that are defined through the choice of the VKC. These extra degrees of freedom offer hints for investigating extra constraints that may have to be added. Alternatively, the planner may decide that a particular degree of freedom aligns with an axis of symmetry and should *not* be considered for extra constraints.

The feature function approach represents the remaining degrees of freedom only implicitly, in the null space of the feature Jacobian. If this null space is one-dimensional, i.e. the rank of the feature Jacobian is 5, then this degree of freedom can be derived numerically: A twist that is projected into the null space yields the null space's direction – only the sign of the resulting movement is undefined and can be fixed by an suitable heuristic. However, when the null space is of higher dimension, then this naive approach is problematic, because there is no inherent unique basis for the null space. It is more promising, to add constraints to the given set, akin to the removal of redundant constraints in the previous section. This requires a ranking of potential constraints that may be added to the constraint set. Tentatively adding one constraint at a time, it is checked whether the new constraint increases the rank of the feature Jacobian matrix. This procedure continues until the feature Jacobian has full rank. Possible candidate constraints may also be derived from a virtual kinematic chain that otherwise would have been chosen based on object and tool symmetries.

4.1.4.3 Taking ranges into account when evaluating redundancy

The conflict-checking approach described so far does not take into account the ranges for the constraints. When a constraint is currently fulfilled, it is switched off, so another 'conflicting' constraint could be active at this moment. Considering this, it is well possible to push more than six constraints into a tool-object relationship while having a perfectly working controller. No part of the presented system prevents such a setup. However, in this situation, the choice of constraint ranges becomes interdependent and therefore more difficult to handle. If two constraints cannot be unified using e.g. interval arithmetic, as proposed in Section 3.2.3, the algorithm is extended easily for checking such constraint sets for 'actual' conflicts: For any given object-tool pose, it considers only the constraints that would be active, i.e. that are violated. The rows of the interaction matrix, which belong to deactivated constraints are removed. If this modified interaction matrix is rank deficient, then there are (at least) two active conflicting constraints.

4.1.5 End-effectors with limited Dexterity

One assumption for the above definition of dependency is that the transform between object and tool can be adjusted freely, for instance by grabbing the tool or the object with a robot arm. Consider in contrast the use of a robot elbow: The framework allows to place the end-effector at the elbow of the robot, for example to close a drawer. Since the elbow is directly 'attached' to the robot shoulder it can only be controlled by three shoulder axes. Typically, this only allows to move the elbow on a sphere, while the third degree of freedom influences the orientation of the elbow and cannot be exploited for most tasks. Thus, this end-effector is unable to fulfill many constraints that would be considered feasible by the above definition. In this case, the robot must consider the elbow's movement capabilities, which are encoded in the robot Jacobian:

$$\text{rank}(\mathbf{J}_F \mathbf{J}_R) \quad (4.5)$$

By multiplying with the robot Jacobian, I ensure, that the robot is able move in the necessary directions and fulfill the requested constraints. Note that in the case of an elbow closing a drawer, not even the elbow position can be controlled freely. But the allowed height and lateral position ranges leave enough freedom for successful execution. Therefore, this constraint set is only deemed valid, if the allowed constraint ranges are taken into account.

The previous approach from Equation 4.4, which only uses the feature Jacobian, can be seen as a simplification where $\mathbf{J}_R = \mathbf{I}$, i.e. a Cartesian robot is assumed for the robot arm.

4.1.6 Discontinuity Detection

We can now detect, whether a set of constraints is dependent for all relative poses of the involved objects. But how can we detect if a feature function has a discontinuity for a particular pose? For example, the function `pointing_at` is undefined when the tool is on top of the object – just as one can only go south at the north pole. In the following, I approach this problem numerically, by computing the second derivative of the feature function at a critical pose. Then I explore, how critical poses might be

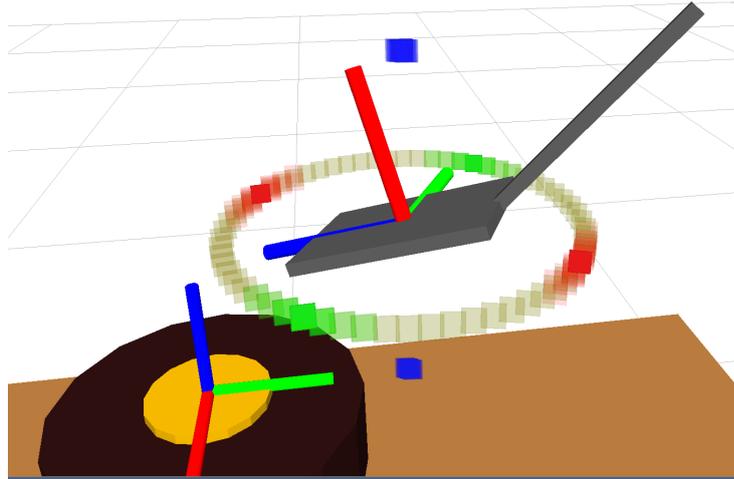


Figure 4.15.: *Discontinuity plot for Euler Angles: When all coordinate axes align with a marker of their respective color, then a discontinuity is hit*

found, either by sampling the space of rigid transformations, generating 'degenerate' transformations like rotations of 90° , or by analyzing the structure of my particular feature functions.

To detect a discontinuity, I compute the second derivatives or curvature in every coordinate direction, using three equidistant samples ($\mathbf{h}_{-i}\mathbf{x}$, \mathbf{x} , $\mathbf{h}_i\mathbf{x}$) of the task function f around the current pose:

$$\frac{\partial^2 f}{\partial^2 x_i} \approx \frac{f(\mathbf{h}_{-i}\mathbf{x}) - 2f(\mathbf{x}) + f(\mathbf{h}_i\mathbf{x})}{h^2} \quad (4.6)$$

Where \mathbf{h}_1 through \mathbf{h}_3 are small finite translations and \mathbf{h}_4 through \mathbf{h}_6 are small finite rotations. The matrices with a negative index are negative translations and rotations. Their computation is explained in section A.4. The value of h is the step size of these translations and rotations.

Discontinuities have infinite curvature, so a sufficiently small h yields arbitrarily high curvature when a singularity is hit. However, the smaller h is, the more precisely must the discontinuity be found. In practice, if f'' exceeds a given threshold, I assume to have found a discontinuity.

This method allows us to check if a discontinuity is present in a region of size h around a given (relative) pose \mathbf{p} . But how can I find a critical pose? I can obviously

4. Execution of movements

sample the 6D space of rigid transformations in steps of h . But this can be too time-consuming. During execution, the current pose is relevant, since a discontinuity may threaten the stability of the controller. But at the planning stage, the equivalent – a perturbation-free simulation run – may be sub-optimal.

In order to increase the chances of catching a discontinuity, I sample special 'degenerate' transformations, like rotations of 90° , zero translation, translation along single axes and combinations thereof. For example, sampling all 24 rotations where the tool is axis-aligned, plus an arbitrary rotation around one of the x- y- or z- axes is a relatively small search space, but covers the singularities of all Euler Angle representations (see Figure 4.15). Like exhaustive sampling, this method is applicable to all task functions, regardless of their inner structure.

For my feature functions, I determine relevant poses by the function's construction: For example, `pointing_at` is a 'Cos' function of two vectors. When either of these vectors is of zero length, then a discontinuity might appear. More formally, for a feature function of the form $\text{Cos}(v_1, v_2)$, the cases $\text{Len}(v_1) == 0$ and $\text{Len}(v_2) == 0$ are of interest for a discontinuity check. While these conditions do not directly yield relative object poses for a test, they can be used in a gradient-descent-like algorithm in order to control a simulated, Cartesian robot towards such a critical pose.

For the example of `pointing_at`, two critical cases arise: (1) The tool is on/over the object and (2) the tool is pointing down or up. In both cases, the intuitive definition of 'looking from above, where is the spatula pointing' is undefined. Different from the feature function `perpendicular`, the function `pointing_at` appears to have the same singularities as RPY angles.

Thus, the singularity problem for the function `pointing_at` is not better than for RPY. But in these singular poses, the perpendicular constraints are still usable. The problems have been isolated in only one feature function in which they reveal themselves to exist by construction: The (2D-) direction is simply not defined above the object. By deactivating only the critical `pointing_at` constraint, the task can still be executed successfully.

4.1.7 Evaluation

In this section I compare the Virtual Kinematic Chains and Feature Functions with each other and examine their fitness to perform alignment tasks. Again, I use pancake-flipping as an example task.

The question might arise, why the RPY or Euler angle representation was chosen in the first place in the iTaSC framework. Indeed, there are at least 12 Euler-Angle representations, and it is known that any such representation must have singularities of some sort, due to the different topology of the space of rotations SE_3 and the Euclidean space \mathbb{R}^3 . Aside from the angle-wraparound at the angle 2π , they all have some singularity where the same orientation can be represented by many different angles. When dealing with such a singularity in an Euler Angle representation, some iTaSC controllers must even switch to a different variety of Euler angles when approaching a singularity. One reason is that a Virtual Kinematic Chain, as used in the iTaSC framework, must cover the six degrees of freedom of 3D position and orientation with exactly six parameters. This requires a non-redundant representation of angles. Unit Quaternions use four numbers to represent a rotation and have a unit constraint on their norm. This would introduce an extra constraint on the task angles that is not modeled in the framework, which complicates the use of Quaternions or any other redundant angle representation.

Another reason is that I want to assign semantics to the angle parameters. In a sense, both VKCs and Feature Functions perform a Coordinate Transformation from twists in Cartesian space into task-relevant movement direction. Using Quaternions on the task-space side defeats this purpose, because I cannot assign a meaning to each of the Quaternion parameter numbers. So dealing with singularities appears to be unavoidable. But which of the many Euler angle representations is useful for a particular task? Which one does not have a singularity in the middle of a movement? How must the reference frame be arranged? In order to resolve these questions in a automated way, a formal description of the desired semantics is required, one that can be matched against one Euler angle parameter.

Consider again the pancake flipping task. In this example I have manually chosen RPY angles and aligned them such that roll turns around the spatula's symmetry axis, pitch turns around the front edge and yaw turns around the blade normal. For this

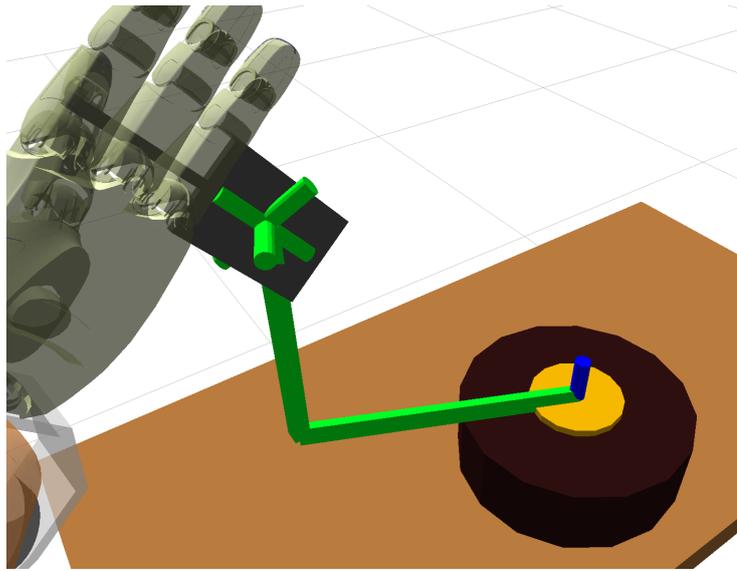


Figure 4.16.: *A non-trivial orientation where RPY-angles loose their meaning.*

parametrization it seems easy to assign semantics to the axes: roll aligns the front edge with the oven, pitch aligns the side edge and yaw determines the direction in which the spatula is pointing. But this intuition is only true when at least two of the three angles are zero: The 'pointing' semantics actually implies a rotation around the oven normal, rather than the blade normal. Also the axes that change one edge alignment without affecting the other, are neither fixed with respect to the spatula, nor with respect to the oven. Not surprisingly, the absolute values of these angles loose their meaning, too, when I consider non-trivial orientations. This is particularly bad since these are parameters that the planner should determine through reasoning. Consider the example pose in Figure 4.16. The spatula is rotated such that both the front and the side edge are at roughly 45 degrees to the oven plane, pointing about 20 degrees left to the oven center.

The feature-based constraints were set up to represent exactly these alignments, so they correctly report these values as $(a_f=-39^\circ, a_s=50^\circ, p=18^\circ)$. The RPY angle representation loses this meaning: It reports $(a_f=-17^\circ, a_s=-18^\circ, p=-73^\circ)$. While deviations in a_f and a_s might be tolerable in certain situations, the pointing direction has lost meaning completely. At the $(0,0,0)$ -orientation, this meaning was still valid.

The Euler angles are a general-purpose orientation representation which has requirements that are not always present for a particular task. For example, one usually

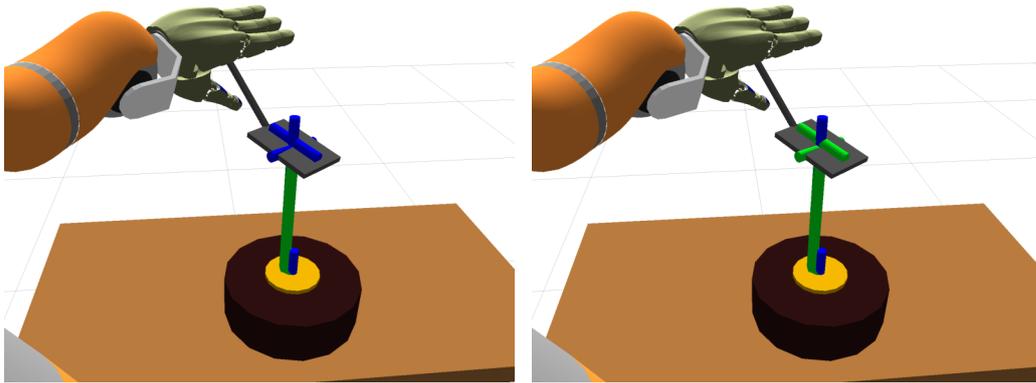


Figure 4.17.: Constraints at singular positions (left: RPY, right: feature-based). The green rotation/translation axes are fine, the blue axes have a discontinuity and are thus unusable for control

wants to distinguish between turning left and turning right – a reasonable requirement in itself. But this in turn requires to define a 'point-of-view', so 'left' and 'right' can be distinguished. This was illustrated in Section 4.1.2. This distinction is not always necessary and may lead to extra, representational singularities.

To show this, I now compare the virtual linkage constraints with the feature-based constraints near a singularity: The spatula is moved over the center of the oven. At this place, two angles are undefined: 1) the direction where the spatula is coming from – it is already there – and 2) the direction in which the spatula is pointing (relative to the center), it is always 'away' from the center. However, the alignment constraint for the front-edge and the side-edge of the spatula can be understood independently. The following constraints become uncontrollable:

approach	angle	dist	height	align-front	align-side	point-at
virtual-linkage	X	.	.	X	X	X
feature-based	X	X

Table 4.3.: The constraints that are controllable over the center of the oven

For the virtual-linkage solution, the angles depend on the cylinder coordinates: At the singularity at $\text{dist}=0$, all three angles become discontinuous. For the feature-based constraints, the align-front and align-side constraints are completely independent of the position and thus remain well-defined and controllable.

4. Execution of movements

In addition to being easier to translate from symbolic specifications and the more direct grounding in the scene, the Feature Function approach avoids several singularities that otherwise must be dealt with using heuristics, like adding extra constraints or changing angle representations when approaching a singularity.

I thus got rid of some constraints that I had to define using the virtual kinematic chain like:

- minimum distance to center
- z-axis must not point down or up (Euler-singularity, see Figure 4.15)

Some singularities remain, inherent in the problem specification: There is no way to define a pointing direction towards the center when one is already at the center. Such singularities are hard to avoid completely at specification time, but at least they can be detected automatically at planning time, so that they can be deactivated, or, if necessary, extra constraints can be synthesized to avoid them.

4.1.8 Conclusion

In this section I have introduced the Virtual Kinematic Chains and Feature Functions and have evaluated their use and automated construction for every-day activities: I have demonstrated probabilistic comparison in the counting of possible kinematic chains and a similar method was used to compare feature functions and to detect conflicts between them. I introduced a general visualization method for all task functions and demonstrated the advantages of the more modular and fine-grained Feature Functions over Virtual Kinematic Chains, which have better grounding in the task description and reduced dependencies between each other.

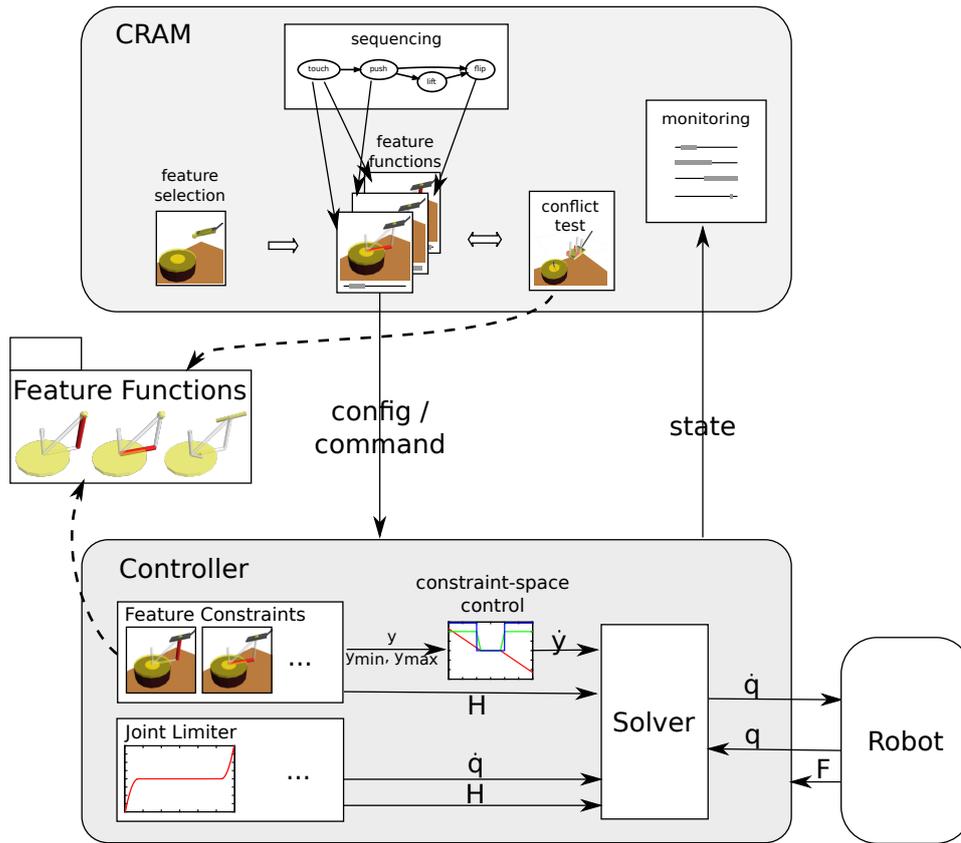


Figure 4.18.: Overview of the control architecture

4.2 Controller Architecture

After defining all geometric features, feature functions, and desired ranges, a suitable controller must be created that can successfully execute such movements. It is the controller's task to fulfill the promise, that these specifications can be used as control laws without further manual intervention.

Figure 4.18 shows the structure of our control architecture: The planner defines the action sequence and constructs constraints, which consist of a *feature function* which relates two *geometric features* to each other and shall be controlled to a *desired range* of values. This constraint set is checked for conflicts using the feature function library, before it is sent to the controller.

The controller uses the same feature functions to compute their current value and the interaction matrix \mathbf{H} . According to the desired value ranges, it computes task-space

4. Execution of movements

velocities \dot{y} which are sent to the solver. Using the interaction matrix, the solver then converts these task-velocities into suitable joint velocities which are sent to the robot. In the null space of this Feature Constraint Controller, secondary tasks like joint limit avoidance are executed.

The next sections discuss the general control equations that are used by the solver and explain the task-space velocity generation based on the desired ranges. I then describe the solver's null space projection and present several null-space tasks. I conclude with an experiment that demonstrates the exploitation of the robot's null space.

4.2.1 Control Scheme

As discussed in Section 2.4.1, I choose a controller over a planning approach, because the robot needs to react to force interaction (which, by its nature, is control). Further requirements were that compositions of several constraints can be executed, disturbances are handled 'correctly' and remaining degrees of freedom can be exploited.

There is an approach that integrated constraints into a probabilistic planner: In this way, position constraints can be fed to the planner and suitable plans can be found, even in cluttered environments. But the force interactions can not be simulated beforehand, due to unavoidable uncertainty in the robot's perception.

My control framework is based on the task function approach. Recall that a task function is a differentiable function of the robot's posture. Using its derivative, the robot is controlled such that the task function yields a desired value. This is a very powerful approach which can combine a large variety of constraints in a unified way. It is possible to define a task function over different sensor data like the measurements of a force-torque sensor. But the derivative must still be defined over the robot's posture. This derivative contains the information how the robot can control these measurements.

In particular, I take the approach used in the iTaSC framework: This framework does another abstraction and defines task functions over the poses of objects and tools, some of which are attached to the robot end-effector. This gives us the ability to

abstract from the concrete robot or seamlessly use two-arm manipulation. Just as for the discussion of the feature-constraint representation, I relate two objects to each other, (at least) one of which must be attached to a robot arm. This object is called tool. Please note that this is only intended to make the formulas easier to understand. Clearly, the roles of tool and object can be swapped, or both can be attached to robot arms, as will be discussed shortly. Let the pose of the tool be \mathbf{x}_t and the pose of the object be \mathbf{x}_o . The relative pose between those two objects is $\mathbf{x} = \mathbf{x}_o^{-1}\mathbf{x}_t$. The task function that scores the relationship between the two objects is $\mathbf{y} = \mathbf{f}_t(\mathbf{x})$. Differentiating the task function with respect to time yields:

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \dot{\mathbf{x}} \quad (4.7)$$

The matrix $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is called the interaction matrix \mathbf{H} and $\dot{\mathbf{x}}$ becomes the twist \mathbf{t} of the tool with respect to the object:

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{t} \quad (4.8)$$

By applying the correct transformations, the robot Jacobian \mathbf{J}_R relates joint velocities to the Cartesian velocity (or twist) of the end-effector or – by applying the correct transformations – the twist of the tool.

Assuming that the object is static, we can write

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{J}_R\dot{\mathbf{q}} \quad (4.9)$$

But also movements in the object or both tool and object can be modeled directly: Remembering that the relative pose \mathbf{x} is composed as $\mathbf{x} = \mathbf{x}_o^{-1}\mathbf{x}_t$, we can similarly decompose the twist \mathbf{t} into $\mathbf{t} = \mathbf{t}_t - \mathbf{t}_o$. We can now independently compute the \mathbf{t}_t and \mathbf{t}_o using the Jacobians of the robots, to which they are attached

$$\dot{\mathbf{y}} = \mathbf{H}(\mathbf{J}_{Rt}\dot{\mathbf{q}}_t - \mathbf{J}_{Ro}\dot{\mathbf{q}}_o) \quad (4.10)$$

Equivalently, the Jacobians and joint velocities can be stacked to yield

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{J}_R\dot{\mathbf{q}}_R \quad (4.11)$$

4. Execution of movements

$$\text{with } \mathbf{J}_R = \begin{bmatrix} \mathbf{J}_{Rt} & -\mathbf{J}_{Ro} \end{bmatrix} \text{ and } \dot{\mathbf{q}}_R = \begin{bmatrix} \dot{q}_t \\ \dot{q}_o \end{bmatrix}$$

Please note that in order to achieve consistent control, the Jacobian matrices need to be transformed to have the same *reference frame* and *reference point*. Usually, Jacobians are computed so that the reference frame is aligned with the robot base and the reference point is the end-effector. The reference frame can be changed using the so-called *screw projection matrix* and the reference point is change using the *reference point transformation*. Both are described in Section A.1. By expressing movement constraints in an object-centric way (rather than w.r.t. robot joints) the specific robot and a one-arm or two-arm configuration are abstracted away.

In my implementation, I exploit the data structures available in a ROS-based robot to construct the robot specific part: The 'URDF' description (Universal Robot Description Format) of the robot contains the kinematic and dynamic structure of base, arms, torso and head. The 'tf' transformation tree, is constructed from this description and the current joint angles and is augmented with all known tool and object poses. This information allows to construct \mathbf{J}_R with minimal effort.

The interaction matrix \mathbf{H} , that is, the derivative of the task function ($\mathbf{H} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}$), can often be found analytically. However, the computational power available today easily allows to compute \mathbf{H} numerically:

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}_i} = \frac{\mathbf{f}(\mathbf{h}_i \mathbf{x}) - \mathbf{f}(\mathbf{x})}{h} \quad (4.12)$$

where $\mathbf{h}_{\{1,2,3\}}$ are translations of step size h in x -, y - and z -direction, respectively, and $\mathbf{h}_{\{4,5,6\}}$ are rotations of step size h around the x - y - and z axis, as described in Section A.4.

Using this numerical computation, new elementary constraint functions can be added more easily. For more computationally expensive task functions, the derivative can also be provided directly.

The joint velocities, that are commanded to the robot are computed by inverting the combined interaction- and robot Jacobian matrix

$$\dot{\mathbf{q}} = (\mathbf{H}\mathbf{J}_R)^+ \dot{\mathbf{y}} \quad (4.13)$$

where $(\mathbf{HJ}_R)^+$ is the Moore-Penrose Pseudo Inverse, which is calculated using a singular value decomposition. Inverting both matrices jointly not only saves time but allows the task to 'ignore' otherwise troublesome singularities of the robot: For example, during the pancake flipping, the robot aligns its wrist towards the hand singularity, in order to make the turning movement faster. This singularity means, that the hand can no longer move around one axis in space. However, the task does not require this movement, which is encoded in the interaction matrix. Thus, no fast joint movement are even attempted.

However, in order to avoid other unwanted robot movements, the *damped least squares* pseudo inverse is used, where (near singularities) a factor λ trades joint velocities against tracking accuracy.

When the inverse is computed in this way, a particular solution is chosen which optimizes $\|\dot{\mathbf{q}}\|$, i.e. the mean squared norm of joint velocities. This norm weights all joints equally, the base axes (with the whole robot attached to it) and the hand axes, with only the tool attached. A more meaningful norm arises, when the robot mass matrix is taken into account. Then, the kinetic energy of the robot is minimized. This is implemented using a *weighted damped least squares* pseudo inverse $(\mathbf{HJ}_R)^\#$, as explained in Section A.5.

This weighted pseudo inverse also offers to weight the error on the task side. This parameter is harder to find general, since the unit of the task directions can be length, angle or, possibly, an arbitrary unit. Luckily, this question is only important when the robot is under-actuated compared to the task (otherwise, the error can be eliminated). But in this work, the focus is on exploiting all possible freedoms in tasks using human-inspired 7-DOF arms. An over-constrained movement is detected ahead of time and can be modified before the robot tries to execute it. However, the desired ranges of every constraint direction, provide a hint as to how critical it is to fulfill particular constraint. This information could be exploited in order to construct a suitable task-space weighting matrix. But the problem remains that the robot behavior becomes very hard to predict.

4.2.2 Range Controller

So far, the control Equation 4.11 provides a relation between the (change of the) feature function values and robot joint velocities. In some sense, it is a coordinate transformation between the task-relevant feature functions and robot kinematics.

In the next step, I specify the dynamics of the task functions, i.e. how the task function values \mathbf{y} shall change over time in order to fulfill the constraints. Recall, that the desired values for the task function are expressed as value ranges, which allows us to specify both equality and inequality constraints in the same framework. These range constraints can be interpreted as two inequality constraints. They can be solved elegantly with more specialized methods as in (Decré et al., 2009). However, for this application, a simpler approach was sufficient. The goal is to find a controller that moves towards fulfilling these range constraints while leaving as much freedom as possible to the robot for optimization. This yields two basic requirements: When a constraint is violated, i.e. the current value of a task function is outside the desired range, then a velocity towards that range should be computed. When a constraint is fulfilled – the current value is inside the desired range – then the robot is free to move inside this range and the respective constraint direction should be ignored and should not influence the robot movement.

At this level, an elementary constraint is represented using the desired position- and force ranges:

$$([\mathcal{y}_{lo}, \mathcal{y}_{hi}], [F_{lo}, F_{hi}])$$

where $[\mathcal{y}_{lo}, \mathcal{y}_{hi}]$ is the range of desired positions and $[F_{lo}, F_{hi}]$ is the range of desired forces. When a constraints shall be deactivated temporarily, both ranges can be set to $[-\infty, \infty]$.

For every constraint axis with the current position y and the allowed range $[\mathcal{y}_{lo}, \mathcal{y}_{hi}]$ I implement the position constraint as a P-controller with a 'dead zone' inside the

allowed range. When y is inside the allowed range, I lower the weight w of the constraint so that a lower level tasks can exploit that direction:

$$\dot{y} = \begin{cases} K_p(y_{hi} - m - y) & : y > y_{hi} - m \\ K_p(y_{lo} - m - y) & : y < y_{lo} + m \\ 0 & : \text{otherwise} \end{cases} \quad (4.14a)$$

$$w = \begin{cases} 0 & : y_{lo} + m < y < y_{hi} - m \\ 1/m(y - y_{hi}) + 1 & : y_{lo} < y < y_{lo} + m \\ 1/m(y_{lo} - y) + 1 & : y_{hi} - m < y < y_{hi} \\ 1 & : \text{otherwise} \end{cases} \quad (4.14b)$$

Where y is the current constraint value, \dot{y}_p is the desired change of the constraint value, K_p is a plant-dependent control coefficient and m is a margin by which the controller shall push inside the allowed range. This is visualized in Figure 4.19. The constraint weight w defines whether this constraint direction shall be controlled at the moments. All constraint weights are composed into a diagonal matrix \mathbf{W} , which is multiplied to the left of the interaction matrix

$$\dot{\mathbf{y}} = \mathbf{W}\mathbf{H}\mathbf{J}_R\dot{\mathbf{q}} \quad (4.15)$$

This approach has the advantage that when all constraints are fulfilled, then the null space, is not restricted in any way, since all task weights are zero. So, in addition to the directions that are redundant for the given task, like the symmetry axis of the pancake, also the task-relevant movement directions can be exploited to some degree. This gives unprecedented flexibility for optimization, be it for efficiency, smoothness or dexterity. Several approaches are presented in section 4.2.3.

The presented P-Controller, has the property that all constraints arrive at their desired ranges at the same time, since the commanded velocity is scaled with the distance-to-go. However, in practice this simple approach has several flaws: When a constraint is far from it's desired range, the computed velocity and acceleration can become very high. But by simply limiting task velocities and -accelerations independently, the synchronous execution is lost. In the current implementation, this task is handled by the Reflexxes library (Kroger and Wahl, 2010) which yields reasonably smooth trajectories.

4. Execution of movements

Using the free implementation of this library, I obtain trapezoidal velocity curves with limited acceleration. For fast reaching movements, a bell-shaped velocity curve may be closer to human movements ((Flash and Hogan, 1984)). But humans also constantly perform a more global optimization for jerk, effort and precision. Such an optimization would be outside the scope of this thesis. However, the presented constraints provide the frame in which such an optimization may take place: The constraints that must be fulfilled throughout a movement phase define the space of allowed trajectories and the constraints that should be fulfilled at the end of a movement phase are the goal states. In the scope of this research, the presented generator was sufficient to demonstrate the power of the feature-constraint based movement representation.

By the impedance control of our robots, position and force are coupled by the configured stiffness. Therefore, we can apply a force by simply offsetting the position. The required offset can be computed using the controller's spring constant K_F

$$\Delta y_f = K_F(F_{desired} - F_{measured}) \quad (4.16)$$

In my implementation, the position range has priority over the force range. This limits the robot's potential movements to a well known range and prevents the robot from 'running away' when an expected contact does not occur. An unexpected collision is handled by limiting the length of the robot's virtual spring. If a collision has occurred and the controller tries to move the *commanded* robot position beyond a defined

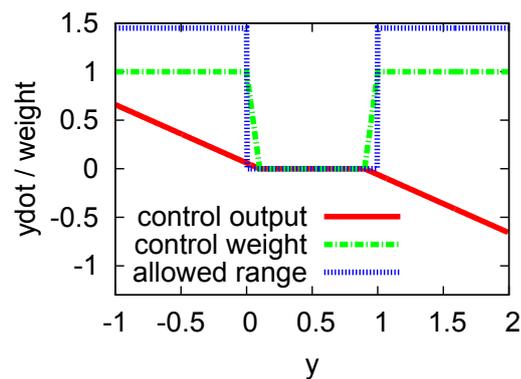


Figure 4.19.: Controller that keeps a task angle inside a predefined range.

maximum distance from the current position, then that movement is limited to the allowed range.

The force has lower priority: If a constraint is inside its position range but outside its desired force range, then an offset is added to the position controller, until either the desired force is reached or the allowed position range is left. This effectively narrows the range of desired (commanded) positions.

For collision avoidance, this strategy makes the robot softer as long as it is inside its position range. When the position range is left, the defined robot stiffness applies. For the case of applying a contact force, the robot moves into its desired position range and on in the same direction until a desired force is measured. But the robot will not move past the end of the position range.

Obviously, these strategies are overlaid with the configured robot stiffness, but they have worked very well in practice. A more direct approach is to modulate the robot's Cartesian stiffness matrix which is presented in Section 4.3.

4.2.3 Null Space Exploitation

Throughout this thesis, it was stressed repeatedly that a minimal constraint representation should leave as much freedom to the robot as possible. This freedom should be used for optimization. In this section I will explain four optimizations and how they are projected into the null space.

Recall that the robot joint velocities for the primary task and $\dot{\mathbf{q}}_1$ were computed as

$$\dot{\mathbf{q}}_1 = (\mathbf{W}\mathbf{H}\mathbf{J}_R)^+ \dot{\mathbf{y}}_1 = \mathbf{J}_1^+ \dot{\mathbf{y}}_1 \quad (4.17)$$

where $\mathbf{J}_1 = \mathbf{W}\mathbf{H}\mathbf{J}_R$ is the combined Jacobian of the primary task, $\dot{\mathbf{y}}_1$ is the desired velocity in task space.

The null space projection is done using Equation 4.18

$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \dot{\mathbf{y}}_1 + (\mathbf{I} - \mathbf{J}_1^+ \mathbf{J}_1) \dot{\boldsymbol{\xi}} \quad (4.18)$$

4. Execution of movements

where $\dot{\xi}$ is a secondary joint velocity that shall not affect the main task. The term $(\mathbf{I} - \mathbf{J}_1^+ \mathbf{J}_1)$ projects $\dot{\xi}$ into the null space of \mathbf{J}_1 . Now I want to find $\dot{\xi}$ such that a set of secondary constraints is fulfilled, which is specified in terms of another task Jacobian \mathbf{J}_2 and desired task velocities $\dot{\mathbf{y}}_2$

$$\dot{\mathbf{y}}_2 = \mathbf{J}_2 \dot{\mathbf{q}} = \mathbf{J}_2 (\mathbf{J}_1^+ \dot{\mathbf{y}}_1 + (\mathbf{I} - \mathbf{J}_1^+ \mathbf{J}_1) \dot{\xi}) \quad (4.19)$$

solving for $\dot{\xi}$ yields

$$\dot{\xi} = (\mathbf{I}_1 - \mathbf{J}_1^+ \mathbf{J}_1)^+ \mathbf{J}_2^+ (\dot{\mathbf{y}}_2 - \mathbf{J}_2 \mathbf{J}_1^+ \dot{\mathbf{y}}_1) \quad (4.20)$$

Inserting this into 4.18 yields the control equation

$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \dot{\mathbf{y}}_1 + (\mathbf{I} - \mathbf{J}_1^+ \mathbf{J}_1) (\mathbf{I}_1 - \mathbf{J}_1^+ \mathbf{J}_1)^+ \mathbf{J}_2^+ (\dot{\mathbf{y}}_2 - \mathbf{J}_2 \mathbf{J}_1^+ \dot{\mathbf{y}}_1) \quad (4.21)$$

This procedure can be applied recursively for an arbitrary number of hierarchical levels. In order to improve stability, the damped least-squares pseudo-inverse is used (Chiaverini, 1997). In the development of the ‘‘Stack of Tasks’’, a lot of research has been dedicated to dealing with singularities, see e.g. (Keith et al., 2011; Keith, 2010). For the experiments in this thesis, the damped least-squares pseudo-inverse was adequate.

The secondary task Jacobian \mathbf{J}_2 and desired task velocities $\dot{\mathbf{y}}_2$ can be derived in the same way as the primary ones: By specifying a task function which is differentiated numerically. For secondary tasks which are specified in joint space, the task space is obviously the robot joint positions and the task Jacobian becomes the identity \mathbf{I} . However, when the null space of such sub-tasks shall be exploited in a third hierarchy level, a weighting matrix like in 4.14 is required to appropriately reduce the rank of \mathbf{J}_2 .

After introducing the null projection, I introduce three possible ways to exploit the task null space. Firstly, two general sub-tasks are presented to avoid joint limits and optimized dexterity using the so called manipulability ellipsoid. I then demonstrate how the robot’s work space is extended significantly, by specifying a manipulation task using constraints. After that, I explain how the task execution can be smoothed by executing the next movement in the null space of the current movement. Finally, I

propose to optimized the movement speed for the next movement by optimizing the manipulability in the direction of the next movement.

4.2.4 Joint Limit Avoidance

The first null space task presented here is the avoidance of joint limits. While this task may not seem very spectacular, it has proven to be essential to ensure robust robot operation in changing environments.

The joint limit avoidance task is designed to fulfill two simple requirements: (1) A joint limit is only avoided when the joint angle is within a distance of d from the limit, (2) the maximum repelling velocity is h . Furthermore, the function is continuous and differentiable over the whole joint range.

In order to fulfill these requirements, a piecewise quadratic function was chosen which is depicted in Figure 4.20:

$$\dot{q}_a = \begin{cases} \frac{h}{d^2}(q_{lo} - q)^2 & : q < q_{lo} + d \\ -\frac{h}{d^2}(q_{hi} - q)^2 & : q > q_{hi} - d \\ 0 & : \text{otherwise} \end{cases} \quad (4.22)$$

Where q_{hi} and q_{lo} are the upper and lower joint limits, respectively.

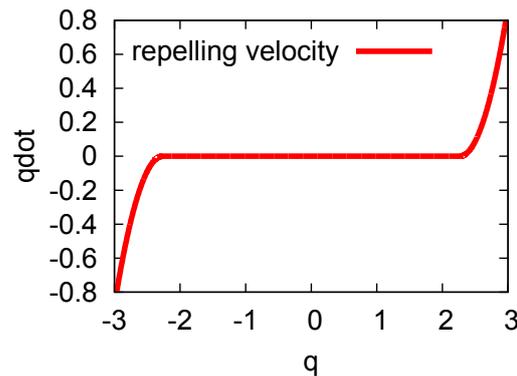


Figure 4.20.: Behavior of the joint limit avoidance.

With the null space projection scheme from Section 4.2.3, I use these velocities as task velocities $\dot{\mathbf{y}}_2$ and the identity as the task Jacobian $\mathbf{J}_2 = \mathbf{I}$. Obviously, all joints which are more than d away from their limits could be ignored. It is possible to reduce an axis weight like in the range-based controller from section 4.2.2 by reducing the respective diagonal entry of \mathbf{J}_2 . This would allow to combine joint limit avoidance with other tasks. But in the experiments, it has been used as the lowest level task, so a weighting was not necessary.

There are more capable methods for joint limit avoidance in (Marchand et al., 1996) or (Chaumette and Marchand, 2001) but this method was sufficient for the given task.

4.2.5 Manipulability Optimization

Besides the distance from joint limits, the so-called manipulability ellipsoid is a common measure of dexterity (Yoshikawa, 1984). It is a local linear approximation of how fast a robot can move in a given spatial direction with limited joint velocities. This ellipsoid is computed from the robot Jacobian \mathbf{J} as $\mathbf{J}\mathbf{J}^T$. Yoshikawa defined a 'manipulability measure' as

$$w = \sqrt{\det(\mathbf{J}\mathbf{J}^T)} \quad (4.23)$$

which is proportional to the volume of said ellipsoid. The shape and orientation can be recovered by means of a singular value decomposition of $\mathbf{J}\mathbf{J}^T$:

$$\mathbf{J}\mathbf{J}^T = \mathbf{U}\mathbf{D}\mathbf{V}^T \quad (4.24)$$

The entries $\sigma_1, \sigma_2, \dots$ of the diagonal matrix \mathbf{D} represent the squared principal axis lengths of the ellipsoid and \mathbf{U} represents its rotation.

Optimizing for manipulability requires to find the gradient of w :

$$\nabla w = \frac{\partial \sqrt{\det(\mathbf{J}\mathbf{J}^T)}}{\partial \mathbf{q}} \quad (4.25)$$

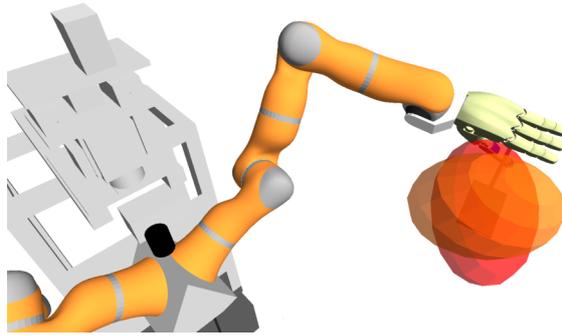


Figure 4.21.: *Manipulability- and Force Ellipsoid*

This gradient can be used as the desired velocities of a secondary task as $\dot{\mathbf{y}}_2 = \nabla w$. Since this task function is defined on the joints, I can again use the identity as the Jacobian matrix $\mathbf{J}_2 = \mathbf{I}$.

4.2.6 Optimize for next movement

Another option is to run the next movement in the subspace of the current movement. All non-conflicting constraints of the next movement will be fulfilled as well. All conflicting movements will at least move the corresponding primary constraint to the right border of it's desired range, in preparation for the next move.

The major issue for this approach is obtaining the next movement from the planner – it might be that it depends on the outcome of the current movement or on a perception task that is not yet done.

Another challenge is the optimal timing at which the subsequent movement should be activated to produce 'natural' movements where the secondary tasks finish in sync with the primary task. A particularly noteworthy work is (Keith, 2010), where the scheduling of such an overlapping task execution is investigated.

4.2.7 Optimize for speed of next movement

If it is not possible to perform a part of the next move, then it might be possible to prepare the robot kinematics, such that the next movement can be performed faster.

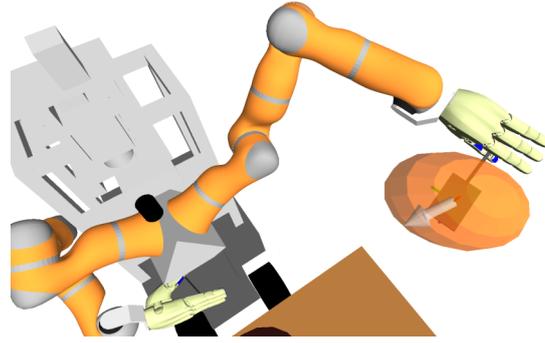


Figure 4.22.: *The (translational) Manipulability Ellipsoid, evaluated in a particular direction*

This is also useful, when a movement was annotated to be done 'quickly'. The turning movement of the pancake flipping is a good example. In order to do this, the robot must find the main movement direction of the next constraint set and optimize the manipulability ellipsoid to extend further in this direction.

In order to find the main movement direction of a constraint set, the robot evaluates it at the current robot postures and computes the would-be task velocities using the simple, non-clamped p-control law from Equation 4.14. This creates zero velocities for already fulfilled constraints while all non-fulfilled constraints have a velocity proportional to their distance-to-go. Using the numerically computed interaction matrix \mathbf{H} , these hypothetical secondary task velocities $\dot{\mathbf{y}}_{next}$ are transformed into the twist that the robot would (like to) execute:

$$\mathbf{t}_{next} = \mathbf{H}^+ \dot{\mathbf{y}}_{next} \quad (4.26)$$

For convenience, assume that this vector is scaled to length 1 for the following derivation.

Given the direction of the next movement, we need to evaluate the manipulability ellipsoid in the direction of \mathbf{t}_{next} . Consider the quadric surface of all points \mathbf{x} that satisfy

$$\mathbf{x}^T \mathbf{J} \mathbf{J}^T \mathbf{x} = 0 \quad (4.27)$$

This is an ellipse with the same orientation as the manipulability ellipsoid but reciprocal extents, i.e. $1/\sigma_1, 1/\sigma_2, \dots$ etc. This shape is also called the force ellipsoid: Longer extents express that the robot can exert more force in that direction with given joint

torque limits. For the manipulability ellipsoid we must invert $\mathbf{J}\mathbf{J}^T$ and replace it in equation 4.27:

$$\mathbf{x}^T(\mathbf{J}\mathbf{J}^T)^{-1}\mathbf{x} = 0 \quad (4.28)$$

Finding the size in the direction of \mathbf{t}_{next} amounts to re-scaling \mathbf{t}_{next} with a factor $\lambda \in \mathbb{R}$ and solving the equation

$$\lambda \mathbf{t}_{next}^T (\mathbf{J}\mathbf{J}^T)^{-1} \mathbf{t}_{next} \lambda = 0 \quad (4.29)$$

This yields the solution

$$\lambda = \frac{1}{\sqrt{\mathbf{t}_{next}^T (\mathbf{J}\mathbf{J}^T)^{-1} \mathbf{t}_{next}}} \quad (4.30)$$

By taking the gradient of this solution, the robot can optimize for fast execution of the next movement or (by omitting the inversion), for stronger forces in the movement direction:

$$\dot{\mathbf{y}}_2 = \frac{\partial \lambda}{\partial \mathbf{q}} \quad (4.31)$$

as this task is a joint space optimization, we again omit the computation of it's Jacobian (which could be done numerically) and assume $\mathbf{J}_2 = \mathbf{I}$.

4.2.8 Evaluation: Extension of Work Space

One goal of the proposed movement representation is to increase the robot's work space by allowing it to exploit the task's null space. In order to evaluate this advantage I repeatedly tested the pancake-flipping task in simulation.

I generated 30 random poses for the oven in a 0.3 x 0.3 x 0.4 meter space and executed the pancake flipping once with the constraint based controller and once using fixed relative poses between the oven and the spatula which were taken from one trial of the constraint based controller. This was executed by activating all six constraints and setting them to the recorded positions, leaving no room for optimization. Since this scene is rather static, the same performance could have been achieved by planning suitable poses in advance. A randomized planning algorithm which can take into account the same kind of constraints is presented in (Jäkel et al., 2010) or (Berenson et al., 2009). The results are shown in Table 4.4.

4. Execution of movements

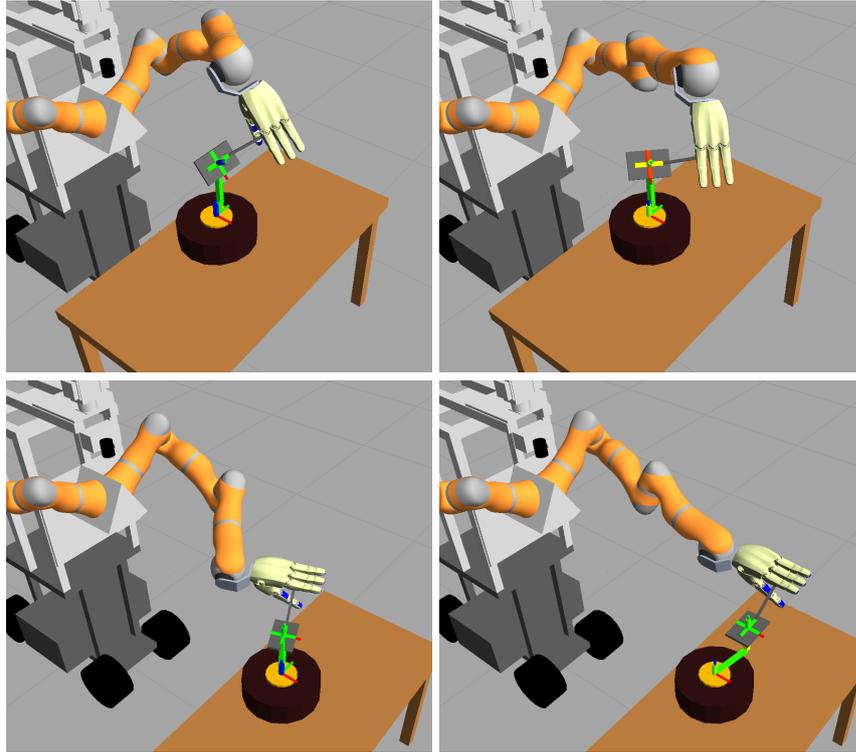


Figure 4.23.: *Snapshots of the workspace evaluation (left: constraint based approach, right: pose based approach)*

	approach	under	lift	flip	total
constraint-based	30	30	30	15	115
pose-based	21	23	12	0	56

Table 4.4.: *Success statistics for the pancake-flipping task*

The constraint based controller can reach more than twice as many poses than the pose based control. Only the flipping pose was posing occasional problems for the new approach. Figure 4.23 illustrates why the constraint based approach was more successful: When the pose-based controller has to stretch the robot arm in order to reach its sideways approach pose, the constraint based controller exploits the symmetry of the oven and chooses a more dexterous and convenient pose.

4.3 Implementation of Force Constraints

Section 3.3 has tackled the specification of desired forces or force ranges for various reasons such as alignment, maintaining contact, or avoiding damage to the environment. It has discussed the advantage of specifying a desired range for both force and position. Even if the task demands pure force- or position control, the other range can act as a 'safety net' to avoid damage to the environment.

This chapter tackles the translation of such force constraints into a desired impedance. With this approach I exploit the proven stability of an impedance controlled robot. This control law guarantees *passivity*, even under unknown, hard contacts.

This approach stands in contrast to the well-known approach of applying pure position control in one direction and pure force control in another direction.

In the rest of this chapter, I first describe, how the classical position/force control can be modeled in our impedance controlled robots. For this approach I exploit an "additional force" term, that our lightweight robots can apply. Then, I evaluate how a desired stiffness can be determined for every task direction, i.e. every dimension of our task function. Naturally, a higher stiffness allows for higher precision while a low impedance lets the robot exploit the environment when adjusting it's pose. After that, I present a way to translate several desired stiffness values into a Cartesian stiffness matrix and how to recover a pivot point, orientation, translational- and rotational stiffness from the resulting stiffness matrix.

4.3.1 Relationship between Force/Torque Control and Impedance Control

A common way to implement hybrid position/force control is to set up some directions with force control and some other (perpendicular) directions with force or torque control, either in a parallel control structure (Raibert and Craig, 1981), or in a cascaded structure with an inner position control loop and an out force control loop (De Schutter and Van Brussel, 1988b; Perdereau and Drouin, 1993; De Schutter et al., 1997). Some sensor is attached to the end-effector which measures the actual force

4. Execution of movements

f_{actual} in the relevant task direction y . For a velocity-resolved robot, the task-level control law is

$$\dot{y} = K_p(f_{desired} - f_{actual})$$

where $f_{desired}$ is the desired or commanded force and \dot{y} is the desired change of y . The task velocities of all constraints \dot{y} are then translated using the robot Jacobian and an interaction matrix which relates velocities in Cartesian space to velocities in task-space:

$$\dot{y} = \mathbf{HJ}\dot{q}$$

This controller basically adjusts the current robot position to exert the desired amount of force. For this controller, the robot position is not limited and may diverge into unwanted areas. In unknown environments with uncertain sensor information this is undesired behaviour and must be monitored on a higher control level.

The steady state of this controller is always $f_{desired} = f_{actual}$ and K_p can be understood as a *damping* coefficient (see also (Mason, 1981)).

In contrast, impedance control is an explicit coupling between *position* and force. The force f is linearly coupled to the deviation of the position through a *stiffness* K :

$$f = K(x_{desired} - x_{actual}).$$

To put it in a nutshell, the force control law relates a force to a velocity and the impedance control law relates a force to a position. While these equations may look very similar, their behaviour is fundamentally different: while the impedance control law couples the actual robot position to a force that it shall exert, the force control law from above moves the robot to whatever position is necessary to achieve the desired force. Depending on the environment, the robot may do arbitrarily large movements.

4.3.2 Approximating Position/Force Control using Impedance Control

Although, impedance control propagates the coupling between force (or torque) and position, it can be tuned to be pure force- or position control.

A very simple way to translate hybrid position/force controllers into impedance terms, is to set the stiffness in the force-controlled directions to zero – and therefore decouple position and force – and set a desired additional force in the same direction.

The KUKA lightweight robot can receive such an additional force. It has a slightly extended control law

$$F = Kx + D\dot{x} + F_{add} \quad (4.32)$$

in this form, I can set K and D to zero and simply apply the desired force F_{add} . This would not be possible without the extra term F_{add} . This term allows to specify a force independently of the current position x . One could, however, control x to maintain the desired force in an outer control loop with a nonzero stiffness. Specifying pure position control can be achieved by setting the stiffness K high. It is known that a very high stiffness can be a challenge for impedance controlled robots, but our robot can be as stiff as 8000 Nm/rad.

When considering several task directions, this approach amounts to setting the stiffness either very high or zero for any particular direction. The force vectors would then be added up to a resulting additional force.

For a Cartesian stiffness matrix \mathbf{K} , a zero stiffness constraint for a particular task direction \mathbf{t} translates into the equation

$$\mathbf{0} = \mathbf{K}\mathbf{t}_f \quad (4.33)$$

i.e. any movement along the task direction should be met with zero opposing force. This way, only the additional force F_{add} has an effect in this task direction.

On the other hand, a stiffness of K_{hi} in the task direction \mathbf{t}_p translates into the equation

$$K_{hi}\mathbf{t}_p = \mathbf{K}\mathbf{t}_p \quad (4.34)$$

i.e. an opposing force/torque that is scaled up with K_{hi} shall be exerted in the task direction \mathbf{t}_p . This is an eigenvalue formulation and it's solution is considered in Section 4.3.4.

4.3.3 Deriving a desired Stiffness

Section 3.3 has specified tasks involving force constraints using a position range and a force range.

I chose to use Impedance Control to implement this behaviour, because of the intuitive coupling between position and force and the stable implementation. For instance, when the table, to which the robot should make contact, breaks down, then a force controller would move the robot down to the floor or to the joint limit, whichever is hit first. An Impedance Controller will simply stop at a somewhat lower position, depending on its programmed stiffness.

This section derives – for every task direction – a stiffness from the desired position and force ranges. It considers how small or large desired force and position ranges influence the choice of the stiffness.

A high stiffness results in a strong coupling between position and force, leading to small position deviations but strong forces when perturbed. A low stiffness, on the other hand, results in a weak coupling between positions and forces, leading to large position deviations but small forces. In this context, perturbations are not only external forces, but also the robots internal friction that is not entirely compensated.

The properties of a high stiffness are:

- + Good position tracking
- \pm Force is easily controlled with only small position changes
- – Position uncertainties and disturbances cause big forces.

The properties of a low stiffness are:

- – Bad position tracking
- \pm Force is adjusted with large position changes.
- + Position uncertainties have little impact on forces.
- + The environment is exploited for alignment tasks.

The allowable position deviations and forces are described in the force- and position ranges and can be used to derive a suitable stiffness.

4.3.3.1 Stiffness for Free-Space Movements

As pointed out in Section 3.3, a free-space movement that cannot collide does not require any compliance. It is only useful when a contact or collision is expected during the movement or when the robot is in contact with the environment.

The case of no (or unexpected) contact is simple: The allowed force range will be symmetric around zero and is the expected force during a collision-free movement. The stiffness K can be scaled such that the maximum position deviation Δp is matched by the maximum allowed force Δf :

$$K = \frac{\Delta f}{\Delta p} \quad (4.35)$$

The relation is outlined in Table 4.5.

	Δp high	Δp low
Δf high	K low	K high
Δf low	K low	–

Table 4.5.: *Qualitative translation of position and force ranges into a stiffness.*

For a large allowed position deviation and a small force range, a low stiffness is appropriate and for small allowed position deviations and large allowed forces, a high stiffness is required. For small allowed position deviation but large allowed forces, a high stiffness is more appropriate: Perturbations have only a small effect on the trajectory and the robot 'pushes' to achieve its position goal. When both, the position range and the force range are large, the robot can choose between low stiffness, where very low force is exerted in case of a collision. In exchange, the position deviations are significant. A higher stiffness allows the robot to control the arm with more precision and optimize in the null space of the task. This corresponds to lowering the allowed force Δf or lowering the allowed position deviation Δp . When both ranges are (very) small, then the robot may not be able to perform such a movement: The robot has some remaining internal friction that is not compensated. This internal fric-

tion will be indistinguishable from external forces that demand a deviation, which in turn would violate the desired small position deviation. Depending on the stiffness, the robot would always violate either the force or position constraint. A very low stiffness may also be beyond the robot's capabilities: The robot would not overcome its internal friction and won't move at all. It would merely comply with its environment and never reach the end of its movement. The trade-off between position-accuracy and stiffness must be taken into account, in order to achieve the required precision of a movement's pose.

The required position accuracy Δp translates into a lower limit on the stiffness that can be set. This limit depends on the internal friction of the robot and friction with the environment. But even when no position control whatsoever is required and stiffness $K = 0$ is sent to the robot, then the internal friction of the robot puts a lower limit to the robot's actual stiffness. Section 4.3.6.2 explores this relationship in an experiment.

4.3.3.2 Stiffness for Contact Formation

The case of an expected contact is slightly more interesting: Here, the desired force range is shifted by a desired contact force. Let the expected contact position be in the range $[c_{lo}, c_{hi}]$. In order to maintain the contact, a contact force in the range $[F_{lo}, F_{hi}]$ is desired. In order to achieve this contact force, the *commanded* position p_{cmd} is moved in the direction of the contact force and may leave the range of expected contact positions. Under disturbances, the actual position may follow up to this commanded position. The stiffness K relates the force range to a position (-offset) range:

$$P_{offset} = \left[\frac{F_{lo}}{K}, \frac{F_{hi}}{K} \right]$$

which is added to the range of expected contacts. By adjusting K , the size of this range and thus the size of the allowed position deviation can be determined. Depending on the direction of contact, either c_{lo} or c_{hi} is replaced with a suitable extension as depicted in Figure 4.24. The resulting range is the position range that must be allowed to the robot.

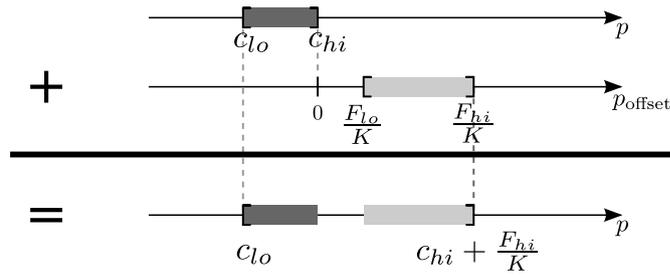


Figure 4.24.: *Illustration of force- and position ranges with a contact force*

This range-calculation reveals, that this calculated position range alone does not carry enough information to compute a stiffness: The size of this interval is the sum of the unknown range of expected contact and the commanded position offsets caused by applying the contact force. If the 'contact range' is assumed to be small, then a larger position offset and thus a lower stiffness is allowed. A large contact range requires a higher stiffness. This ambiguity can be resolved by (1) transmitting the range of expected contacts, or assuming a standard range, (2) transmitting a stiffness or assuming a standard stiffness or (3) detecting the actual contact online and re-adjusting the stiffness accordingly.

As discussed, the allowed position deviation must be larger than the range of the expected contact. For the sake of parametrizing the robot, the allowed position deviation is required, but for monitoring and evaluation of the perception, the range of expected contact is more interesting.

4.3.3.3 Stiffness for Alignment Tasks

A third case of stiffness configuration is the alignment task from Section 3.3, which demanded for zero-torque axes. This is configured by simply setting the respective stiffness to zero, knowing, that the performance may be limited by the robot's hardware. However, this setting may lead to drift in the robot's pose due to inaccuracies of the dynamic model of the tool. This is especially apparent for robots with very good torque sensing. Usually the tool's weight can be measured in a moment of rest, just before contact is made. For measuring the center of gravity, as well, some more measurements are required. A convenient linear algorithm to compute this calibration is given in (Prats, 2009).

4. Execution of movements

Finally, when moving perpendicular to a contact, then the friction at the contact can disturb the movement and lead to larger position deviations at a low configured stiffness. Compared to the considerations of the free-space movement, a systematic error in the position tracking is introduced. Of course, when friction coefficient and contact force are known, then this error can be compensated by adjusting the goal pose. The friction can be measured by observing the position deviation. In order to compensate for the arm-internal friction, a second measurement under a different contact force can be made. But that friction varies with the configured stiffness, as will be shown in Section 4.3.6.2. More generally, these experiments yield estimates for the minimum stiffness that must be configured to reach a certain accuracy.

4.3.4 Composing a Stiffness matrix from Feature Constraints

After assigning a desired (or standard-) stiffness to all feature-constraint axes, the problem arises, how they should be assembled into a stiffness matrix \mathbf{K} . Given a position error \mathbf{t}_i in a task-direction that was specified by a constraint, and the desired stiffness k_i in that direction, then this amounts to an Eigenvalue problem:

$$\tau_i = k_i \mathbf{t}_i = \mathbf{K} \mathbf{t}_i \quad (4.36)$$

where \mathbf{t}_i is the Eigenvector and k_i is the Eigenvalue. For exactly six linearly independent directions, this can be solved as

$$\mathbf{K} = \mathbf{V} \mathbf{D} \mathbf{V}^{-1} \quad (4.37)$$

where \mathbf{V} is a matrix whose columns are the axis directions \mathbf{t}_i and \mathbf{D} is a diagonal matrix containing the stiffnesses.

However, for vectors that are not orthogonal, this yields non-symmetric matrices (real spectral theorem), unless the stiffnesses of the non-orthogonal vectors are equal.

One difficulty is to choose sensible directions and stiffnesses to complete the stiffness matrix to a full-rank 6x6 matrix. If the same stiffness is desired for all of this null space, then choosing directions to span the null space is uncritical: Any set of vectors that are perpendicular to the task impedance requirements (all with the same

nominal stiffness) is equivalent - a singular value decomposition of a so composed matrix will find an orthonormal basis for these remaining vectors. But choosing the right pivot point is critical: If it is not compatible with the task, this does affect the stiffness matrix thus the performance.

4.3.5 Recovering Coordinate System from Stiffness matrix

After composing a stiffness matrix that resembles a given set of Feature Constraints, this matrix must be applied on the robot. However, our LWR only accepts a coordinate frame and six stiffness values, rather than a full stiffness matrix. Assuming the task can be expressed in such a 'compliance frame', I now face the task to recover the coordinate frame and stiffness values from a stiffness matrix \mathbf{K} .

When the coordinate transformation is known, then the stiffness matrix can be transformed so that the stiffness values can be extracted from the diagonal:

$$\mathbf{K}' = \mathbf{F}^T \mathbf{K} \mathbf{F} \quad (4.38)$$

Where \mathbf{F} is the twist transformation matrix

$$\mathbf{F} = \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_{\times} \mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix}$$

in which \mathbf{R} is the rotation and \mathbf{p} is the translation.

Recovering \mathbf{R} and \mathbf{p} is a bit more difficult. I tackle the problem that given a 6x6 stiffness matrix, I want to recover reference point, reference frame and translational / rotational stiffnesses. I assume that a diagonal stiffness matrix \mathbf{K}_0 was transformed by \mathbf{T} :

$$\mathbf{K}_0 = \begin{pmatrix} k_{tx} & & & & & \mathbf{0} \\ & k_{ty} & & & & \\ & & k_{tz} & & & \\ & & & k_{rx} & & \\ & & & & k_{ry} & \\ \mathbf{0} & & & & & k_{rz} \end{pmatrix} = \begin{bmatrix} \mathbf{k}_t & \mathbf{0} \\ \mathbf{0} & \mathbf{k}_r \end{bmatrix}$$

4. Execution of movements

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_x \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

So I get a matrix of the form

$$\begin{aligned} \mathbf{K} = \mathbf{T}^\top \mathbf{K}_0 \mathbf{T} &= \begin{bmatrix} \mathbf{R}^\top & \mathbf{0} \\ \mathbf{R}^\top [\mathbf{p}]_x^\top & \mathbf{R}^\top \end{bmatrix} \cdot \begin{bmatrix} \mathbf{k}_t & \mathbf{0} \\ \mathbf{0} & \mathbf{k}_r \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_x \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}^\top \mathbf{k}_t \mathbf{R} & \mathbf{R}^\top \mathbf{k}_t [\mathbf{p}]_x \mathbf{R} \\ -\mathbf{R}^\top [\mathbf{p}]_x \mathbf{k}_t \mathbf{R} & -\mathbf{R}^\top [\mathbf{p}]_x \mathbf{k}_t [\mathbf{p}]_x \mathbf{R} + \mathbf{R}^\top \mathbf{k}_r \mathbf{R} \end{bmatrix} \\ &=: \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \end{aligned}$$

I can recover the rotation matrix \mathbf{R} and \mathbf{k}_t using a singular value decomposition $[\mathbf{R}, \mathbf{k}_t] = \text{eig}(\mathbf{A})$. Note that \mathbf{R} is not unique: It might be (repeatedly) rotated around any of the x- y- or z-axis by 90 degrees. It also may include a reflection, which can be detected by checking $\det(\mathbf{R}) = -1$ and corrected by negating one of the columns or rows of \mathbf{R} .

In order to recover \mathbf{k}_r , I must correct \mathbf{D} for it's translational component. This part can be computed as $\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B}$:

$$\mathbf{CA}^{-1}\mathbf{B} = (-\mathbf{R}^\top [\mathbf{p}]_x \mathbf{k}_t \mathbf{R})(\mathbf{R}^\top \mathbf{k}_t^{-1} \mathbf{R})(\mathbf{R}^\top \mathbf{k}_t [\mathbf{p}]_x \mathbf{R}) = -\mathbf{R}^\top [\mathbf{p}]_x \mathbf{k}_t [\mathbf{p}]_x \mathbf{R}$$

so

$$\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B} = \mathbf{R}^\top \mathbf{k}_r \mathbf{R}$$

and \mathbf{k}_r can be recovered with

$$\mathbf{k}_r = \mathbf{R}(\mathbf{D} - \mathbf{CA}^{-1}\mathbf{B})\mathbf{R}^\top$$

the remaining unknown $[\mathbf{p}]_x$ can be recovered from \mathbf{B} as

$$[\mathbf{p}]_x = \mathbf{k}_t^{-1} \mathbf{R} \mathbf{B} \mathbf{R}^\top$$

since $[\mathbf{p}]_x$ is a matrix of the form

$$[\mathbf{p}]_x = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix},$$

the vector \mathbf{p} can be recovered by

$$\mathbf{p} = \frac{1}{2} \begin{pmatrix} [\mathbf{p}]_{x(32)} - [\mathbf{p}]_{x(23)} \\ [\mathbf{p}]_{x(13)} - [\mathbf{p}]_{x(31)} \\ [\mathbf{p}]_{x(21)} - [\mathbf{p}]_{x(12)} \end{pmatrix}$$

Obtaining \mathbf{p} can also be formulated more directly as finding the point where a pure rotation causes no translational forces:

$$\begin{bmatrix} \mathbf{0} \\ \tau \end{bmatrix} = \mathbf{K} \mathbf{T}_p \begin{bmatrix} \mathbf{0} \\ \theta \end{bmatrix}$$

with the reference point transformation \mathbf{T}_p and the stiffness matrix \mathbf{K}

$$\mathbf{T}_p = \begin{bmatrix} \mathbf{I} & [\mathbf{p}]'_x \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

this leads to the condition

$$\begin{bmatrix} \mathbf{A}[\mathbf{p}]'_x & \mathbf{B} \end{bmatrix} \theta = \mathbf{0},$$

for any θ . This, can be written as $\mathbf{A}[\mathbf{p}]'_x = -\mathbf{B}$ and therefore

$$[\mathbf{p}]'_x = -\mathbf{A}^{-1}\mathbf{B} = (\mathbf{R}^\top \mathbf{k}_t^{-1} \mathbf{R})(\mathbf{R}^\top \mathbf{k}_t [\mathbf{p}]_x \mathbf{R}) = \mathbf{R}^\top [\mathbf{p}]_x \mathbf{R}.$$

Note that here I omitted the computation of \mathbf{R} , so that I obtain the skew-symmetric matrix for the rotated point $\mathbf{R}^\top \mathbf{p}$. After using \mathbf{R} to correct for this rotation, \mathbf{p} can be recovered as above.

4.3.6 Evaluation

This evaluation examines how adapting the robot's stiffness can be useful for contact formation and alignment tasks in a kitchen scenario. It tests the feasibility of the proposed force constraint implementation. It looks at practical performance, estimates contact friction and investigates, how the robot's internal friction can be taken into account.

4.3.6.1 Spatula Alignment

This experiment translates an alignment constraint into a force/torque constraint and evaluates how orientation errors can be corrected with a compliant robot arm. It improves precision and reliability of the alignment which in turn it improves the success rate of the robot.

Using torque-control to achieve alignment is not a new approach and is known to work. However, previous experiments have mostly been conducted with high-fidelity force-torque sensors and rather stiff objects. In this experiment, the robot is using the built-in joint torque-sensors, a soft spatula which in turn is grasped with an impedance-controlled hand. Finally, the lever that is given by half the spatula with, is only about 0.04m long. All these factor raise the question, whether it is still useful to modulate the robot's stiffness, rather than setting a soft default stiffness and simply applying enough force.

When pushing a spatula under a pancake, aligning the front edge of the spatula with the oven is critical in order not to destroy the pancake. As described in Section 3.3, I support this alignment by a zero rotation stiffness axis, which is both perpendicular to the front edge and perpendicular to the oven surface normal, so it is roughly aligned along the spatula main axis. This experiment demonstrates the effectiveness of impedance control for this alignment: Depending on the downward force, a misalignment of up to 10° can be corrected.

The robot repeatedly moves the spatula in its hand down onto oven. Different amounts of misalignment are generated to simulate calibration inaccuracies. The rotational

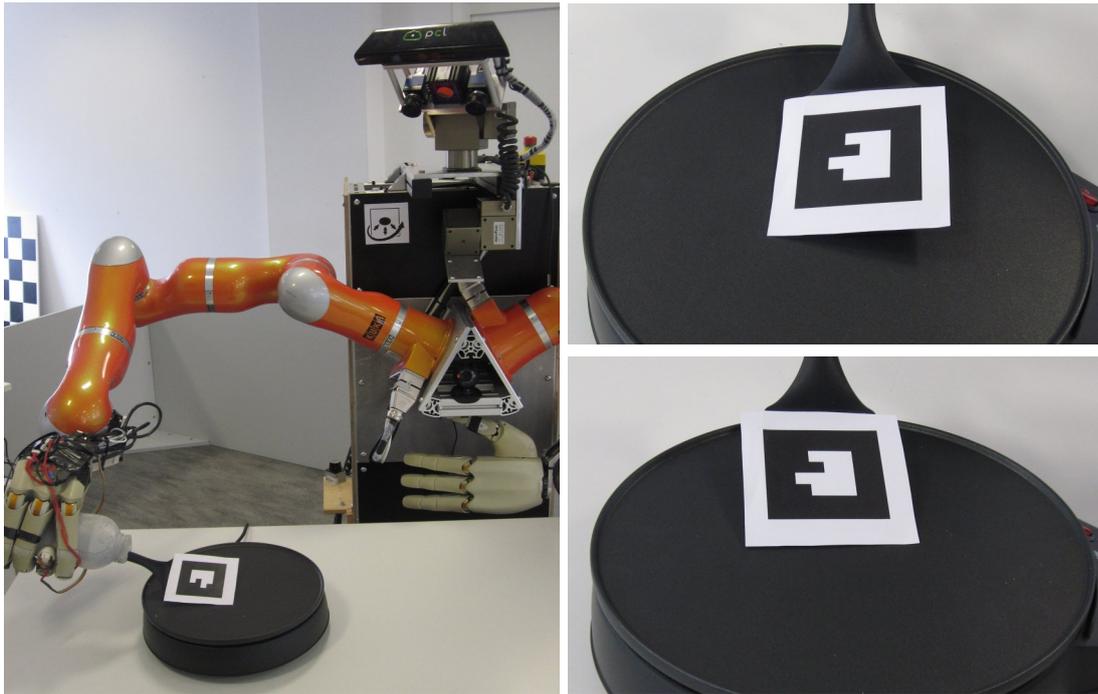


Figure 4.25.: *Experimental setup of the spatula alignment*

stiffness around the forward direction is set to zero and the contact force $F_z = K_z * offset_z$ is varied roughly from 0 N to 45 N by changing $offset_z$.

As a success indicator, a marker on the spatula is tracked with the robot's Kinect sensor which measures the orientation around the y-axis.

The spatula is misaligned by various angles, up to 22 degrees in 4-degree steps. Also the contact force is varied from 0 N to 45 N.

Figure 4.26 shows the effect of the zero-torque constraint: The left diagram shows that the spatula can be aligned with a lower contact force. Or, conversely, with the same contact force, a bigger misalignment can be compensated. It is clearly visible, that also the rotational 'standard' stiffness of 40 Nm/rad can compensate substantial misalignment, also due to the compliance in the spatula tool and the impedance-controlled hand. Even for "no force", the zero torque case yield a better alignment, because for strong misalignment, the corner of the spatula is already touching the surface.

4. Execution of movements

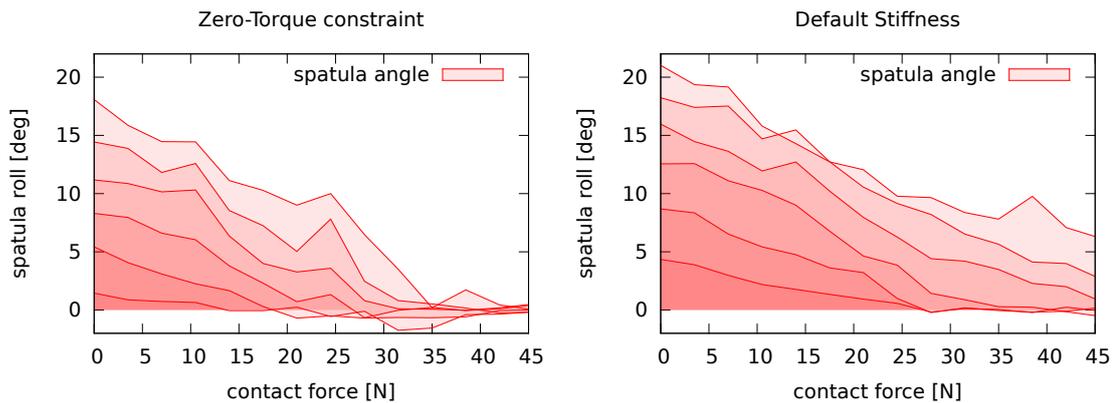


Figure 4.26.: *Effect of a zero-torque constraint for spatula alignment*

4.3.6.2 Spatula Friction

When moving in free space, the robot only has to deal with its joint friction force (and, possibly, a tool or object in its hand). When in contact, the contact friction is added, which (roughly linearly) depends on the contact force.

The following experiment quantifies the effect of internal- and contact friction, and attempts to measure it using Cartesian impedance control.

The spatula is pushed onto the oven and pushed forward with a defined forward stiffness. Both, the contact force $F_{contact}$ and the forward stiffness K are varied. The sliding friction $F_{friction}$ is estimated using the position deviation from the movement goal, i.e.

$$F_{friction} = K \Delta x \quad (4.39)$$

This friction is modeled as Coulomb friction:

$$F_{friction} = \mu F_{contact} \quad (4.40)$$

But the total friction that can be measured is the sum of some internal arm friction and the sliding friction:

$$F_{total} = F_{friction} + F_{internal} \quad (4.41)$$

Thus, to measure both the arm-internal friction and the friction coefficient, at least two measurements are necessary.

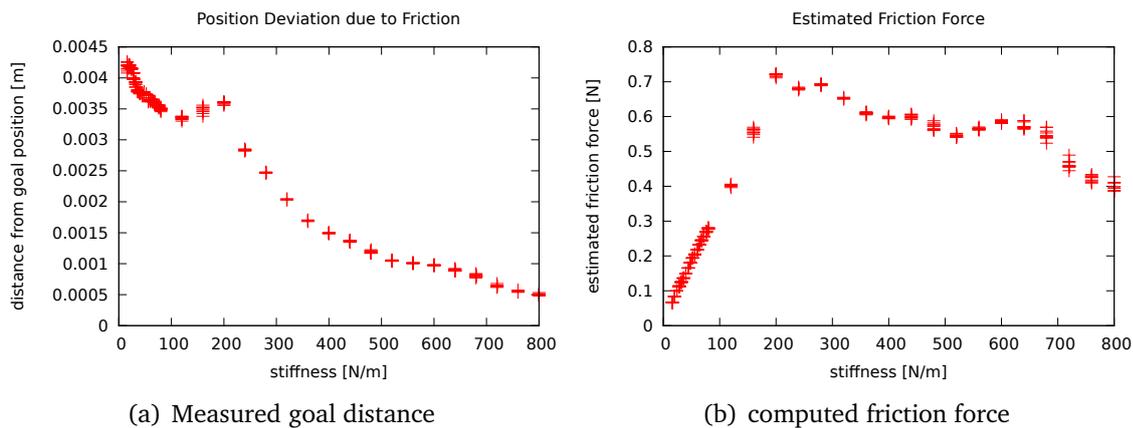


Figure 4.27.: *Measured goal distance and computed friction force for different configured robot stiffnesses*

Figure 4.27 (a) presents some raw data of the pushing experiments for a range of stiffnesses up until 800 N/m. Beyond that stiffness, the measured position deviation dropped below 0.5mm and was no longer relevant for this scenario.

The spatula was *not* in contact with the surface, thus the experiment measured the internal friction of the arm. The position deviation varied widely from day-to-day so that it is difficult to model it in advance. However, the experienced friction was fairly stable for a single experiment.

Figure 4.27 (b) shows the computed friction, which is around 0.6 N for a stiffness of 200 N/m or above. Also the position disturbance has roughly the expected shape of the function $x \rightarrow 1/x$. Below 200 N/m the position deviation – and thus the computed friction – is much smaller. Obviously, the controller switches to a better friction model for compensation.

Figure 4.28 shows the computation of the friction coefficient. The left graph contains the raw data which shows a relatively small effect on the position deviation. The right graph shows the computation of the friction coefficient (0.02 N/N), which is the slope of the fitted line. This computation was done at a stiffness of 60 N/m, where the difference due to different contact force was more pronounced. The measured internal friction is nearly zero in this experiment.

4. Execution of movements

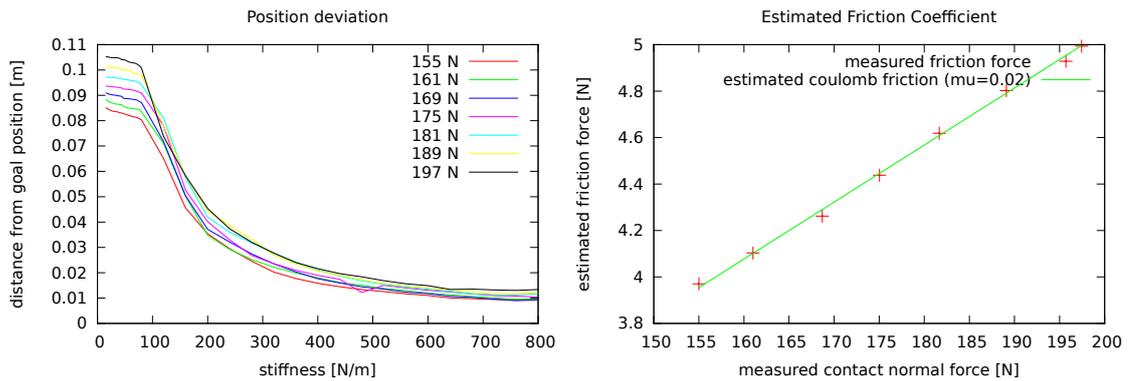


Figure 4.28.: *Measured goal distances and computed friction coefficient for pushing movement*

The data shows that the expected position deviation indeed depends on external (sliding) friction. But in this experiment (and in my experience in kitchen-related demonstrations) this effect is relatively small. The position deviation is mainly a function of the configured stiffness, as Figure 4.28 left shows. These results can only be taken as landmarks, since both the sliding friction may change quickly with the surface, slower movement (Stribeck friction) and environmental changes. Also the robot internal friction is only a coarse estimate and may change over time, too. Nevertheless, these experiments give a hint about a minimum stiffness that should be selected in order to achieve a desired stiffness. They demonstrate that the stiffness can be measured on line which can be used for compensation.

These facts seem to call for an on-line adaptation of the stiffness in order to accommodate for changing friction. But given that the internal friction changes with the configured stiffness and external friction can change quickly, this may lead to instabilities. At least, a strong low-pass filter seems appropriate for this approach.

This thesis has shown a movement description that captures what is important in a movement and what can be changed and optimized. It captures task constraints in a minimal way by using a fine-granular description of movement directions using 1-d task functions and an expressive representation of goals in these directions, using ranges to specify position- or force constraints or both.

The manipulation tasks which I construct with this description, are as constrained as necessary and as relaxed as possible to facilitate 1) optimization and competent reaction to disturbances and 2) automatic adaptation to new objects, tools and environments. Several examples have shown the expressiveness of the representation language.

The constraint-based movement description was able to *tightly couple* the perception, planner and movement execution of the robot: The perception localizes movement-relevant geometric features, the planner assembles them to constraints, which are directly used by the movement execution, which in turn can provide detailed feedback. The description serves as a compact language to exchange information between these components.

This architecture moves more competence to the planner, giving it a very detailed motion description and the information sources to fill in these details.

In order to achieve this integration, the movement description is very *modular*: This thesis has demonstrated various ways to exploit detailed object shape information from perception, bits and pieces from a robotic knowledge-base and trajectory data

5. Conclusion

from demonstrations. Even very small pieces of information could be integrated – the movement representation remains *executable* in every (incomplete) state.

To the best of my knowledge, no other movement description combines the representation of task freedom with this level of detail and modularity.

This thesis has touched a lot of related subjects which may be fruitful to pursue:

One direction of research is to integrate more sensors and perception methods: Additional feature extraction methods such as tool tip detection or superquadric fitting (Kemp and Edsinger, 2006; Sminchisescu et al., 2005) would yield additional geometric features that may describe movement-relevant aspects of objects. For objects or tools that are not rigidly attached to the robot, Virtual Visual Servoing (Comport et al., 2006) could supervise the pose during manipulation actions.

For tactile and force signals, the framework offers a way to categorize transient signals: It knows roughly when the edge of a spatula touches the oven, in what angle and at what speed. This automatically defines supervised learning problems to parametrize a touch- and force sensor based collision detection. Such a detection can help to improve robustness and reduce uncertainty (Kresse et al., 2011; Beetz et al., 2011).

Another interesting question is: How precise must the perception result be? How precise must the orientation of the spatula front edge be? It obviously depends on how the result is used: How much the movement success depends on the spatula's precise orientation, whether impedance control is used to deal with inaccuracies, etc. At least parts of these requirements are stored in the desired position ranges.

It is a challenge that contact forces during manipulation action can hardly be obtained from observation. Therefore, systematically exploring position- and force parametrization (i.e. compliant movement strategies) for their robustness is another obvious task.

Another direction of research is to integrate the constraint-based movement description more tightly with planning and the robot knowledge base:

Since the evaluation of the feature functions in the planner is possible, they could extend the logic-based constraint expressions in the CRAM planner. Conversely, some

of it's constraints could technically be implemented as task functions and used by the controller.

A deeper integration into the robot knowledge base KnowRob is ongoing work (Tenorth et al., 2014): Grounding the semantics of geometric object features, associating constraints with object classes, or merely collecting statistics which feature functions were used on which objects for what tasks.

Another on-going research is on action-effect relationships: This tackles the causal relationship that constraints have on desired and undesired effects in the real world. Humans have an inborn ability to learn, which action has which effect. The idea is to define relatively free exploration tasks to have the robot build up it's world knowledge. If these relationships are stored in a suitable format, they may help assembling new movement descriptions.

This effort goes beyond success and failure: It lets the robot learn action-effect relationships directly, also (and especially) for unwanted effects: Which sort of movement makes the pancake fall down? How does it get destroyed? After which movement is the spatula stuck in the oven? The more movements for undesired effects are known, the better the constraints that avoid these effects.

Finally, more scenarios could be tackled where exploiting the task freedom is advantageous:

For examples, the presented framework naturally extends to two-hand manipulation, which is a 'classical' domain for constraint-based control: For two-hand manipulation, the object-relative pose has 6 dof, both arms have 14 dof. Secondary 'soft' constraints are e.g.: visibility, background, force, manipulability, 'comfort' and many more.

In many two-arm settings, the second hand is used to hold the object: Constraints conveniently describe the set of forces that the robot may (or should) apply to an object – this could be exploited to select a suitable grasp that can hold the object in place.

Tasks like fetching objects or cleaning up could include the robot base in the kinematic chain, so the same algorithms would optimize where the robot stands.

5. Conclusion

This thesis has explored the interesting field of how to 'automate' the creation of a general and transparent constraint-based movement description and getting this description into 'daily use' for 'normal' robot programmers and seemingly simple tasks.

This has yielded a deep and elegant integration between robot software components, and more transparency in the construction of movement descriptions and it sets the frame for later optimizations.

It has also led us to an even more interesting field: A wealth of potential information sources (web, movement data, etc.), several methods to evolve a given movement description and the prospect to integrate with a large class of constraint- and movement descriptions.

This way of combining symbolic and control-theoretic approaches appears to be a promising direction for robot programming and may lead to interesting research questions for years to come.

List of Prior Publications

The work presented in this document is partly based on prior publications. Sections of this work that drew upon content from prior publications cited the respective publications where appropriate. A complete list of publications that were (co-)authored during my research as a doctoral candidate is provided below.

Georg Bartels, Ingo Kresse, and Michael Beetz. Constraint-based Movement Representation grounded in Geometric Features. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2013.

Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

Michael Beetz, Freek Stulp, Piotr Esden-Tempski, Andreas Fedrizzi, Ulrich Klank, Ingo Kresse, Alexis Maldonado, and Federico Ruiz. Generality and Legibility in Mobile Manipulation. *Autonomous Robots Journal (Special Issue on Mobile Manipulation)*, 28(1):21–44, 2010a.

Michael Beetz, Freek Stulp, Bernd Radig, Jan Bandouch, Nico Blodow, Mihai Dolha, Andreas Fedrizzi, Dominik Jain, Uli Klank, Ingo Kresse, Alexis Maldonado, Zoltan Marton, Lorenz Mösenlechner, Federico Ruiz, Radu Bogdan Rusu, and Moritz Tenorth. The Assistive Kitchen – A Demonstration Scenario for Cognitive Technical Systems. In *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Muenchen, Germany, pages 1–8, 2008. Invited paper.

Ingo Kresse and Michael Beetz. Movement-aware Action Control – Integrating Symbolic and Control-theoretic Action Execution. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3245–3251, 2012.

5. Conclusion

Ingo Kresse, Ulrich Klank, and Michael Beetz. Multimodal Autonomous Tool Analyses and Appropriate Application. In *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

Freek Stulp, Ingo Kresse, Alexis Maldonado, Federico Ruiz, Andreas Fedrizzi, and Michael Beetz. Compact Models of Human Reaching Motions for Robotic Control in Everyday Manipulation Tasks. In *Proceedings of the 8th International Conference on Development and Learning (ICDL)*, 2009.

Appendix A

Appendix

In the following sections, various transformation rules are collected that are used throughout this thesis. Then, the weighted damped least squares pseudo inverse is explained. Then, the mathematical foundation is discussed, on which our probabilistic constraint comparison is based.

A.1 Twist Transformation

The coordinate system that is used to express instantaneous spatial velocity (a twist): The directions which define how the direction of translation and translation must be interpreted. But there is also the *reference point* around which pure rotations are performed. Both references can be changed independently using two different transformation matrices. They are defined as follows:

Changing the reference frame can be achieved using the 6x6 *screw projection matrix* \mathbf{P}

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

where \mathbf{R} is a 3x3 rotation matrix. Intuitively, this means rotating both the translation vector and the rotation axis around \mathbf{R} . Changing the reference point is done using the 6x6 *reference point transformation* \mathbf{M} :

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & [\mathbf{p}]_x \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $[\mathbf{p}]_x$ is the 3x3 *skew-symmetric matrix* which is used to represent a cross product as a vector-matrix product $\mathbf{p} \times \mathbf{q} = [\mathbf{p}]_x \mathbf{q}$ with a 3x1 vector \mathbf{p} :

$$[\mathbf{p}]_x = \begin{bmatrix} 0 & -p_z & p_y \\ p_z & 0 & -p_x \\ -p_y & p_x & 0 \end{bmatrix}$$

This transformation computes the translation that is experienced by the new reference point that due to the rotation of the twist. This movement is added to the translation of the twist. The final twist transformation \mathbf{T} is composed by first changing the reference point and then the reference frame:

$$\mathbf{T} = \mathbf{MP} = \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_x \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix}$$

A Jacobian matrix can be transformed using the same transformation matrices, since the columns of a Jacobian matrix are twists as well – the twists that are caused on the robot end-effector when each axis is moved.

These transformation can be found e.g. in (Stramigioli and Bruyninckx, 2001).

A.2 Inverse Twist Transformation

The task function approach requires an interaction matrix \mathbf{H} , which encodes how a given twist affects the task function. This is the *inverse* of a Jacobian matrix and must be transformed differently.

In order to transform a Jacobian matrix \mathbf{J} , with the transformation matrix

$$\mathbf{T} = \mathbf{MP} = \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_x \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix},$$

the matrix \mathbf{T} it is multiplied to the left of the Jacobian

$$\mathbf{TJ}$$

thus its inverse is

$$(\mathbf{TJ})^{-1} = \mathbf{J}^{-1} \mathbf{T}^{-1}$$

So, an inverted Jacobian must be right-multiplied with an inverted twist transformation matrix, which can be found as

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T [\mathbf{p}]_x \\ \mathbf{0} & \mathbf{R}^T \end{bmatrix}$$

The proof is easily done by multiplying the matrices \mathbf{T} and \mathbf{T}^{-1} :

$$\begin{aligned} \mathbf{T}^{-1} \mathbf{T} &= \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T [\mathbf{p}]_x \\ \mathbf{0} & \mathbf{R}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R} & [\mathbf{p}]_x \mathbf{R} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{R}^T \mathbf{R} & (\mathbf{R}^T [\mathbf{p}]_x \mathbf{R} - \mathbf{R}^T [\mathbf{p}]_x \mathbf{R}) \\ \mathbf{0} & \mathbf{R}^T \mathbf{R} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{aligned}$$

A.3 Stiffness Matrix Transformation

Our simplified stiffness controller is

$$\tau = \mathbf{J}^T \mathbf{K} \mathbf{x}_{err} = \mathbf{J}^T \mathbf{K} \mathbf{J} \mathbf{q}_{err}$$

With the Cartesian stiffness matrix \mathbf{K} , the robot Jacobian \mathbf{J} , the Cartesian control error \mathbf{x}_{err} , the joint control error \mathbf{q}_{err} and the commanded joint torque τ . We want to change the reference frame of \mathbf{K} as if we changed \mathbf{J} :

$$\mathbf{J}' = \mathbf{T} \mathbf{J}$$

Thus

$$\tau' = \mathbf{J}'^T \mathbf{K} \mathbf{J}' \mathbf{q}_{err} = (\mathbf{T} \mathbf{J})^T \mathbf{K} (\mathbf{T} \mathbf{J}) \mathbf{q}_{err} = \mathbf{J}^T \mathbf{T}^T \mathbf{K} \mathbf{T} \mathbf{J} \mathbf{q}_{err} = \mathbf{J}^T \mathbf{K}' \mathbf{J} \mathbf{q}_{err}$$

with $\mathbf{K}' = \mathbf{T}^T \mathbf{K} \mathbf{T}$. Thus, the stiffness matrix must be multiplied on *both* sides with the appropriate twist transformation matrix.

A.4 Twist Exponentials

When computing the numerical derivative of a task function, we are faced with the problem of converting a twist \mathbf{t} , into a 4x4 transformation matrix \mathbf{T} , which moves a small increment d in the direction of \mathbf{t} .

Let the twist be defined as $\mathbf{t} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix}$. Then the transformation \mathbf{T} is

$$\mathbf{T} = \exp \left(\begin{bmatrix} [\mathbf{w}]_{\times} & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix} d \right) \quad (\text{A.1})$$

where $[\mathbf{w}]_{\times}$ is the skew-symmetric matrix of \mathbf{w} and $\exp()$ is the matrix exponential, which can be defined over it's power series

$$\exp(\mathbf{X}) = \sum_{k=0}^{\infty} \frac{1}{k!} \mathbf{X}^k \quad (\text{A.2})$$

Equation A.1 can be written more compactly as a 'twist exponential'

$$\mathbf{X} = \exp(\mathbf{t}d). \quad (\text{A.3})$$

This derivation can be found e.g. in (Rutgeerts, 2007). We now define a unit twist \mathbf{t}_i in coordinate direction i as

$$\mathbf{t}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{t}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \dots, \quad \mathbf{t}_6 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix},$$

so that we can construct an finite transformation of 'step size' h as

$$\mathbf{h}_i = \exp(\mathbf{t}_i h), \quad i \in \{1, \dots, 6\} \quad (\text{A.4})$$

Thus, \mathbf{h}_1 , \mathbf{h}_2 and \mathbf{h}_3 are small translations in positive x-, y-, and z direction and \mathbf{h}_4 , \mathbf{h}_5 and \mathbf{h}_6 are small rotations around the x-, y- and z-axes, all with step size h .

Small transformation in the respective negative directions are easily computed by negating \mathbf{t}_i :

$$\mathbf{h}_{-i} = \exp(-\mathbf{t}_i h), \quad i \in \{1, \dots, 6\} \quad (\text{A.5})$$

These transformations are used in the calculation of the first and second numerical derivative of a task function.

A.5 Weighted Damped Least Squares Pseudo Inverse

We use the so-called Moore-Penrose Pseudo-Inverse of a matrix in the solver of our controller and for the probabilistic constraint comparison. Specifically, we use a modification called Weighted Damped Pseudo Inverse. In order to compute the Moore-Penrose pseudo inverse of a matrix \mathbf{A} , we use a singular value decomposition:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

where \mathbf{U} and \mathbf{V} are orthogonal matrices and \mathbf{D} is a diagonal matrix. Then the pseudo-inverse of \mathbf{A} is

$$\mathbf{A}^{-1} = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T$$

The inverse of \mathbf{D} is easily computed by inverting it's diagonal elements, the singular values. Problems arise, when some of these singular values are zero or close to zero, i.e. if \mathbf{A} is rank-deficient. Usually, values near the machine epsilon are treated as zero (and are *not* inverted). But values slightly larger than that threshold still produce big numbers in \mathbf{A}^{-1} , and thus fast robot movements. Instead, for the *damped* pseudo inverse, every singular value σ_j is inverted with a damping factor λ :

$$\tilde{\sigma}_j = \frac{\sigma_j}{\sigma_j^2 + \lambda^2},$$

yielding the matrix $\tilde{\mathbf{D}}$. This factor limits the velocity near singularities, but reduces tracking accuracy in those regions. The damping factor λ must be chosen carefully. This inverse minimizes the squared sum of joint velocities during execution. A more meaningful optimization criterion is the (instantaneous) kinetic energy. This can be achieved by *weighting* the pseudo-inverse matrix (see (Ben-Israel and Greville, 1974)). On the joint side it is weighted with the joint-space mass matrix $\mathbf{W}_q = \mathbf{M}$ of the robot. A task-space weighting matrix \mathbf{W}_y is used to gradually switch some constraints on or off. These matrices are integrated as follows: Both matrices are multiplied to \mathbf{A} before the singular value decomposition:

$$\mathbf{A}' = \mathbf{W}_y\mathbf{A}\mathbf{W}_q$$

yielding a modified \mathbf{A}' . The singular value decomposition then computes

$$\mathbf{A}' = \mathbf{V}'^T \mathbf{D}' \mathbf{U}'$$

These matrices are recombined as

$$\mathbf{A}^\# = \mathbf{W}_q \mathbf{V}' \tilde{\mathbf{D}}' \mathbf{U}'^T \mathbf{W}_y^T$$

which results in the weighted damped least squares pseudo inverse $\mathbf{A}^\#$. See also (Ben-Israel and Greville, 1974).

A.6 Probabilistic Comparison of Functions

The comparison of kinematic chains in section 4.1.1 is called randomized theorem proving and is roughly based on the work in (Schwartz, 1980) which was applied in (Kortenkamp, 1999) for geometric theorem proving using multivariate polynomials. The comparison problem is reduced to testing whether a function is the zero-function $f(x) = 0$ by forming the difference $f(x) = f_1(x) - f_2(x)$. If $f(x)$ is not the zero-function, then every x for which $f(x)$ returns zero must be a root of f . Depending on the number of test values in each dimension, an upper bound for the number of zeros in a given test set is computed. If more test inputs return zero, then that is proof for the identity of the functions. If there are too many potential zeros, then random values are drawn and checked. Knowing that the test domain of the function is (much) larger than the possible number of roots, the probability of mis-classifying two different functions as identical quickly becomes very small with the number of tests. Applying this approach to the virtual kinematic chains, we have functions of $\mathbb{R}^6 \rightarrow \mathbb{R}^6$ which are a product of six matrices. Each matrix contains sine and cosine functions, which have up to four roots in the domain $[-\pi.. \pi]$. However, only one matrix changes per input value: Rotational joints move the end-effector on a circle and translational joints move it along a line. Analogously to (Schwartz, 1980), we may encounter up to $4/N \cdot N^6 = 4N^5$ roots, if we choose test points in an N^6 lattice with $N > 4$. This number arises because a zero of the function under test may only depend on a single input variable and yields zero for all combinations of the other input variables. The probability to hit a zero in this lattice is therefore

$$p_{zero} = \frac{4N^5}{N^6}$$

For $N > 40$ this probability drops below 0.1. For T independent tests, the probability that the functions differ is $(p_{zero})^T$ and quickly becomes very small for only a few tests.

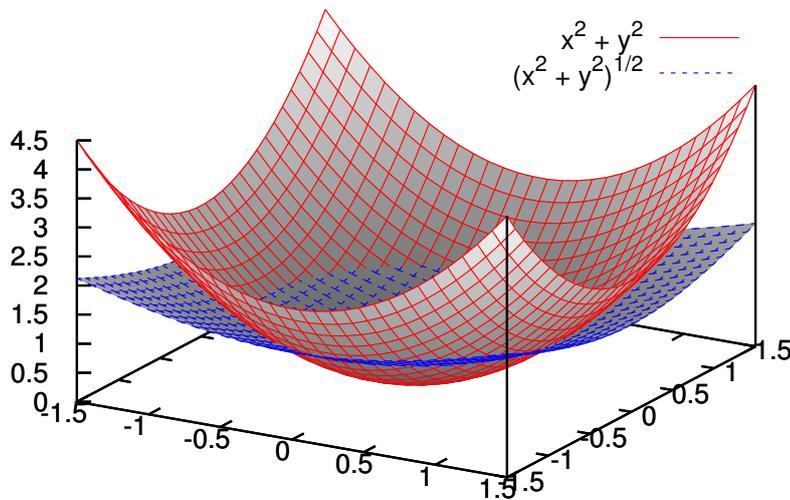


Figure A.1.: *Sample task functions that are different but dependent*

A.7 Task Function Comparison

This section explains the comparison of task functions that is used in section 4.1.4. Two (1D) task functions are deemed identical if the *direction of their gradient* are the same for all poses \mathbf{x} . This test is hidden in the rank computation, which reveals whether the gradients of the two functions-under-test are linearly dependent.

This can also be formulated using the cross product which vanished when both vectors are collinear:

$$\|\text{grad}f_1 \times \text{grad}f_2\| = 0$$

With this formulation we could apply the same method as above. But we want to test much more general task functions of unknown structure. This makes it harder to count the number of possible zeros.

We can reduce the problem by directly estimating the probability that a single 'query' to the function will yield a root: We compare the volume of the domain of f that is covered by the roots to the total volume of the domain:

$$p = \frac{V_{root}}{V_{total}}$$

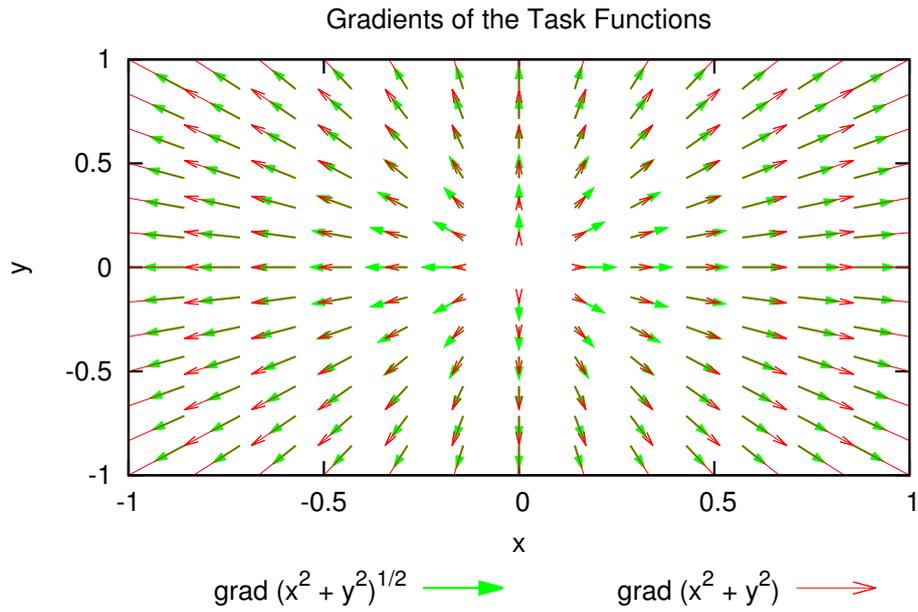


Figure A.2.: Gradients of Sample task functions

Luckily the roots for our kinematic-comparison functions do *not* fill any volume but rather lie on a lower-dimensional manifold. This property is quite inconvenient when equality constraints are used in conjunction with probabilistic planners – these planners *want* to hit these constraints. But here, p becomes very small and a probabilistic proof is valid for almost any (low) number of trials.

Consider, as an example, the functions $f_1 = x^2 + y^2$ and $f_2 = \sqrt{x^2 + y^2}$, which are quite different, as Figure A.1 shows. Consider a controller that can control x and y . This controller can *not* control f_1 and f_2 independently, obviously because their values depend on each other: $f_2 = \sqrt{f_1}$. For any given value of one function, the value of the other function can be computed.

The gradients of these functions are:

$$\text{grad}f_1(x, y) = (2x \ 2y)^\top$$

and

$$\text{grad}f_2(x, y) = \left(\frac{x}{\sqrt{x^2 + y^2}} \quad \frac{y}{\sqrt{x^2 + y^2}} \right)^\top$$

so

$$\text{grad}f_1(x, y) = \alpha(x, y)\text{grad}f_2(x, y)$$

with

$$\alpha(x, y) = 2(x^2 + y^2) = 2f_1(x, y)$$

The gradient of f_1 is a multiple of the gradient of f_2 . It always points in the same direction. Therefore, we can find a scaling term α that expresses one gradient using the other. The sign of this term determines, whether both gradients point in the same or opposite directions.

Consider the controller that controls x and y . When it decides to lower the value of f_1 it will also lower the value of f_2 . These functions can not be controlled independently. Whether the task functions change in the same or opposite directions depends on $\alpha(x, y)$ and is not the decision of the controller.

However, the functions f_1 and f_2 are not the same and behave differently when used as a task function. Their *dynamics* is different, i.e. the velocity (magnitude) which it computes to reach a desired value.

Bibliography

- Erwin Aertbeliën and Joris De Schutter. eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1540–1546, 2014.
- Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.
- Michael Beetz, Moritz Tenorth, Dominik Jain, and Jan Bandouch. Towards Automated Models of Activities of Daily Life. *Technology and Disability*, 22(1-2):27–40, 2010.
- Adi Ben-Israel and Thomas N.E. Greville. *Generalized inverses: theory and applications*, 1974.
- Dmitry Berenson, Siddhartha Srinivasa, David Ferguson, and James Kuffner. Manipulation Planning on Constraint Manifolds. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- H. Bruyninckx and J. De Schutter. Specification of force-controlled actions in the task frame formalism—a synthesis. *Robotics and Automation, IEEE Transactions on*, 12(4): 581–589, 1996.
- S. Calinon, F. Guenter, and A. Billard. On Learning, Representing and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man and Cybernetics, Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007.
- Stephane Cambon, Fabien Gravot, and Rachid Alami. A Robot Task Planner that Merges Symbolic and Geometric Reasoning. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, pages 895–899, 2004.
- F. Chaumette and E. Marchand. A redundancy-based iterative approach for avoiding joint limits: application to visual servoing. *Robotics and Automation, IEEE Transactions on*, 17

(5):719–730, 2001.

Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *Robotics and Automation, IEEE Transactions on*, 13(3):398–410, 1997.

Andrew Comport, Eric Marchand, Muriel Pressigout, Francois Chaumette, et al. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):615–628, 2006.

Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decré, Ruben Smits, Erwin Aertbeliën, Kasper Claes, and Herman Bruyninckx. Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty. *Int. J. Rob. Res.*, 26(5):433–455, 2007.

Joris De Schutter, Dirk Torfs, Herman Bruyninckx, and Stefan Dutré. Invariant hybrid force/position control of a velocity controlled robot with compliant end effector using modal decoupling. *The International Journal of Robotics Research*, 16(3):340–356, 1997.

Joris De Schutter and Hendrik Van Brussel. Compliant robot motion I. A formalism for specifying compliant motion tasks. *The International Journal of Robotics Research*, 7(4):3–17, 1988a.

Joris De Schutter and Hendrik Van Brussel. Compliant robot motion II. A control approach based on external control loops. *The International Journal of Robotics Research*, 7(4):18–33, 1988b.

Wilm Decré, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Extending iTaSC to support inequality constraints and non-instantaneous task specification. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 1875–1882. IEEE Press, 2009.

James Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. Technical report, Stanford University, 2006.

Zachary Dodds, Gregory D Hager, A Stephen Morse, and João P Hespanha. Task specification and monitoring for uncalibrated hand/eye coordination. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1607–1613. IEEE, 1999.

- K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1123–1123. John Wiley & Sons LTD, 1994.
- C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press USA, 1998.
- Richard O. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Technical Report 43R, AI Center, SRI International, 1971.
- Tamar Flash and Neville Hogan. The Coordination of Arm Movements - An Experimentally Confirmed Mathematical Model. Technical Report AIM-786, MIT Artificial Intelligence Laboratory and Center for Biological Information Processing, 1984.
- Anton L Fuhrmann, Rainer Splechtma, and J Přikryl. Comprehensive calibration and registration procedures for augmented reality. In *Immersive Projection Technology and Virtual Environments 2001*, pages 219–227. Springer, 2001.
- Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2002.
- Gregory D Hager and Kentaro Toyama. X vision: A portable substrate for real-time vision applications. *Computer Vision and Image Understanding*, 69(1):23–37, 1998.
- Samad Hayati. Hybrid position/force control of multi-arm cooperating robots. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 82–89. IEEE, 1986.
- Gerhard Hirzinger, N. Sporer, A. Albu-Schäffer, M. Hähle, R. Krenn, A. Pascucci, and M. Schedl. DLR's torque-controlled light weight robot III - are we reaching the technological limits now? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1710–1716, 2002.
- R. Jäkel, S.R. Schmidt-Rohr, M. Lösch, and R. Dillmann. Representation and constrained planning of manipulation strategies in the context of Programming by Demonstration. In *IEEE International Conference on Robotics and Automation (ICRA 10)*, 2010.
- Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1470–1477, 2011.
- François Keith. *Optimization of motion overlapping for task sequencing*. PhD thesis, University Montpellier 2, 2010.

- François Keith, P-B Wieber, Nicolas Mansard, and Abderrahmane Kheddar. Analysis of the discontinuities in prioritized tasks-space control under discreet task scheduling operations. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3887–3892. IEEE, 2011.
- C.C. Kemp and A. Edsinger. Robot manipulation of human tools: Autonomous detection and control of task relevant features. In *Proc. of the Fifth Intl. Conference on Development and Learning*. Citeseer, 2006.
- S.M. Khansari-Zadeh. A Dynamical System-based Approach to Modeling Stable Robot Control Policies via Imitation Learning, 2012.
- S.M. Khansari-Zadeh and A. Billard. Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models. *Robotics, IEEE Transactions on*, 27(5):943–957, 2011.
- O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of*, 3(1):43–53, 1987.
- Hedvig Kjellström, Javier Romero, and Danica Kragic. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding*, 115(1):81–90, 2011.
- Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE, 2010.
- Dipl-Math Ulrich Kortenkamp. *Foundations of dynamic geometry*. PhD thesis, Swiss Federal Institute of Technology Zurich, 1999.
- Ingo Kresse, Ulrich Klank, and Michael Beetz. Multimodal Autonomous Tool Analyses and Appropriate Application. In *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.
- Torsten Kroger, Bernd Finkemeyer, Markus Heuck, and Friedrich M Wahl. Adaptive implicit hybrid force/pose control of industrial manipulators: Compliant motion experiments. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 816–821. IEEE, 2004.
- Torsten Kröger, Bernd Finkemeyer, Ulrike Thomas, and Friedrich M Wahl. Compliant motion programming: The task frame formalism revisited. *Mechatronics & Robotics, Aachen, Germany*, 2004.

- Torsten Kroger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *Robotics, IEEE Transactions on*, 26(1):94–111, 2010.
- Lars Kunze, Mihai Emanuel Dolha, and Michael Beetz. Logic Programming with Simulation-based Temporal Projection for Everyday Robot Object Manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011. Best Student Paper Finalist.
- Lars Kunze, Andrei Haidu, and Michael Beetz. Making Virtual Pancakes — Acquiring and Analyzing Data of Everyday Manipulation Tasks through Interactive Physics-based Simulations. In *Poster and Demo Track of the 35th German Conference on Artificial Intelligence (KI-2012)*, 2012.
- Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Iowa State University, 1998.
- 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 *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6, 2009.
- E. Marchand, F. Chaumette, and A. Rizzo. Using the task function approach to avoid robot joint limits and kinematic singularities in visual servoing. In *Intelligent Robots and Systems '96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, volume 3, pages 1083–1090 vol.3, 1996.
- Éric Marchand, Fabien Spindler, and François Chaumette. ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *Robotics & Automation Magazine, IEEE*, 12(4):40–52, 2005.
- Matthew T Mason. Compliance and force control for computer controlled manipulators. *Systems, Man and Cybernetics, IEEE Transactions on*, 11(6):418–432, 1981.
- Matthew T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.
- Maja J. Matarić. Learning in behavior-based multi-robot systems: policies, models, and other agents. *Cognitive Systems Research*, 2(1):81 – 93, 2001.
- C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira. An introduction to the syntax and content of Cyc. *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question*

- Answering*, pages 44–49, 2006.
- G. Metta, G. Sandini, and J. Konczak. A developmental approach to visually-guided reaching in artificial systems. *Neural Networks*, 12(10):1413 – 1427, 1999.
- Michael Mistry, Peyman Mohajerian, and Stefan Schaal. Arm movement experiments with joint space force fields using an exoskeleton robot. In *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on*, pages 408–413. IEEE, 2005.
- L. Morgenstern. Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking. *Studia Logica*, 67(3):333–384, 2001.
- J Daniel Morrow and Pradeep K Khosla. Manipulation task primitives for composing robot skills. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 3354–3359. IEEE, 1997.
- Lorenz Mösenlechner and Michael Beetz. Parameterizing Actions to have the Appropriate Effects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.
- Lorenz Mösenlechner and Michael Beetz. Fast Temporal Projection Using Accurate Physics-Based Geometric Reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1821–1827, 2013.
- Armin Müller. *Transformational Planning for Autonomous Household Robots using Libraries of Robust and Flexible Plans*. PhD thesis, Technische Universität München, 2008.
- Daniel Nyga, Moritz Tenorth, and Michael Beetz. How-Models of Human Reaching Movements in the Context of Everyday Manipulation Activities. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- Petter Ogren, Christian Smith, Yiannis Karayiannidis, and Danica Kragic. A multi objective control approach to online dual arm manipulation. In *International IFAC Symposium on Robot Control, SyRoCo, Dubrovnik, Croatia*, 2012.
- Véronique Perdereau and Michel Drouin. A new scheme for hybrid force-position control. In *RoManSy 9*, pages 150–159. Springer, 1993.
- Mario Prats. *Robot Physical Interaction through the combination of Vision, Tactile and Force Feedback – Applications to Assistive Robotics*. PhD thesis, Jaume-I University, 2009.

- Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.
- Marc H Raibert and John J Craig. Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):126–133, 1981.
- Johan Rutgeerts. *Constraint-based task specification and estimation for sensor-based robot tasks in the presence of geometric uncertainty*. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, 2007.
- A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157, 2008.
- Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.
- Ken Shoemake. Plücker Coordinate Tutorial. *Ray Tracing News*, 11(1), 1997.
- Cristian Sminchisescu, Dimitris Metaxas, and Sven Dickinson. Incremental model-based estimation using geometric constraints. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(5):727–738, 2005.
- Ruben Smits, Tinne De Laet, Kasper Claes, Herman Bruyninckx, and Joris De Schutter. iTASC: a tool for multi-sensor integration in robot manipulation. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 426–433. IEEE, 2008.
- Ruben Smits, Tinne De Laet, Kasper Claes, Herman Bruyninckx, and Joris De Schutter. iTASC: A Tool for Multi-Sensor Integration in Robot Manipulation. In Hanseok Ko Hernsoo Hahn and Sukhan Lee, editors, *Multisensor Fusion and Integration for Intelligent Systems*, volume 35 of *Lecture Notes in Electrical Engineering*, pages 235–254. Springer, 2009.
- 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 *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2854–2859. IEEE, 2015.
- Louise Stark, Adam Hoover, Dmitry Goldgof, and Kevin Bowyer. Function-based recognition from incomplete knowledge of shape. In *Qualitative Vision, 1993., Proceedings of IEEE Workshop on*, pages 11–22. IEEE, 1993.

- Stefano Stramigioli and Herman Bruyninckx. Geometry and screw theory for robotics. *Tutorial during ICRA, 2001*, 2001.
- Freek Stulp, Ingo Kresse, Alexis Maldonado, Federico Ruiz, Andreas Fedrizzi, and Michael Beetz. Compact Models of Human Reaching Motions for Robotic Control in Everyday Manipulation Tasks. In *Proceedings of the 8th International Conference on Development and Learning (ICDL)*., 2009.
- Moritz Tenorth, Jan Bandouch, and Michael Beetz. The TUM Kitchen Data Set of Everyday Manipulation Activities for Motion Tracking and Action Recognition. In *IEEE International Workshop on Tracking Humans for the Evaluation of their Motion in Image Sequences (THEMIS), in conjunction with ICCV2009*, 2009a.
- Moritz Tenorth, Georg Bartels, and Michael Beetz. Knowledge-based Specification of Robot Motions. In *Proc. of the European Conference on Artificial Intelligence (ECAI)*, 2014.
- Moritz Tenorth and Michael Beetz. KnowRob – Knowledge Processing for Autonomous Personal Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266, 2009.
- Moritz Tenorth and Michael Beetz. A Unified Representation for Reasoning about Robot Actions, Processes, and their Effects on Objects. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. Technical report, IAS group, Technische Universität München, Fakultät für Informatik, 2009b.
- Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1486–1491, 2010.
- Moritz Tenorth, Stefan Profanter, Ferenc Balint-Benczedi, and Michael Beetz. Decomposing CAD Models of Objects of Daily Use and Reasoning about their Functional Parts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5943–5949, 2013.
- Mihran Tuceryan, Douglas S. Greer, Ross T. Whitaker, David E. Breen, Chris Crampton, Eric Rose, and Klaus H Ahlers. Calibration requirements and procedures for a monitor-based augmented reality system. *Visualization and Computer Graphics, IEEE Transactions on*, 1

(3):255–273, 1995.

Nicolai v. Hoyningen-Huene, Bernhard Kirchlechner, and Michael Beetz. GrAM: Reasoning with Grounded Action Models by Combining Knowledge Representation and Data Mining. In *Towards Affordance-based Robot Control*, 2007.

Miomir Vukobratovic and Atanasko Tuneski. Contact control concepts in manipulation robotics-an overview. *Industrial Electronics, IEEE Transactions on*, 41(1):12–24, 1994.

Willow Garage. PR2 Robot, 2008. www.willowgarage.com/pages/robots/pr2-overview.

Daniel Wolpert and Zoubin Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience Supplement*, 3:1212–1217, 2000.

Tsuneo Yoshikawa. Analysis and control of robot manipulators with redundancy. In *Robotics research: the first international symposium*, pages 735–747. Mit Press Cambridge, MA, 1984.