Constraint-based Movement Representation grounded in Geometric Features

Georg Bartels*
georg.bartels@cs.uni-bremen.de

Ingo Kresse†
kresse@cs.tum.edu

Michael Beetz*
beetz@cs.uni-bremen.de

***Abstract*— Robots that are to master everyday manipulation tasks need both: The ability to reason about actions, objects and action effects, and the ability to perform sophisticated movement control. To bridge the gap between these two worlds, we consider the problem of connecting symbolic action representation with strategies from motion control engineering. We present a system using the task function approach [1] to define a common symbolic movement description language which defines motions as sets of symbolic constraints. We define these constraints using geometric features, like points, lines, and planes, grounding the description in the visual percepts of the robot. Additionally, we propose to assemble task functions by stacking 1-D feature functions, which leads to a modular movement specification. We evaluate and validate our approach on the task of flipping pancakes with a robot, showcasing the robustness and flexibility of the proposed movement representation.**

I. Introduction

Robots which are to master everyday manipulation tasks will have to put screws into nuts, push spatulas under pancakes, cut bread into pieces, and so on. Such robots need both: First, the ability to reason about actions, objects, and the effects of actions on objects and second, the ability to perform sophisticated movement control.

Action formalisms in artificial intelligence are designed to reason about actions and their effects. They do, however, abstract away from the way movements are performed. Indeed, reasoning about actions in the context of more sophisticated movements becomes quickly infeasible as has become evident in the context of the well studied "egg cracking" problem [2]. Abstracting away from *how* actions are performed in terms of movements results in actions having non-deterministic effects and the formalisms being incapable of explaining how effects depend on the particular form of movement.

Representations in robot learning [3], [4] and control engineering, on the other hand, specify sophisticated movement control but only in terms of coordinate frames [5] or low-level state and control variables [6]. They typically abstract away from objects and action effects. This is unsatisfactory because in manipulation the robot is to perform a movement to have a desired effect on objects.

To get the best of both worlds, we have investigated the coupling of methods for symbolic action representation in Artificial Intelligence with methods in control engineering in previous work [7]. The goal was to provide formalisms that are strong in expressing *how* movements are to be executed while still allowing the robot to reason about the effects that complex movements might have on objects and scenes.

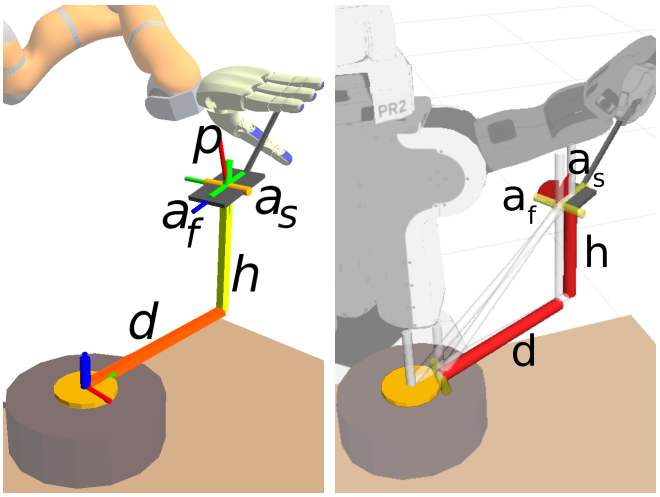


Fig. 1. Two robots making a pancake: (left) related work with restriction to use full frames as control features and a necessity to group constraints in 6-DOF virtual kinematic chains (VKC). (right) the proposed approach with visually-grounded features using 1-D constraint functions.

We modelled motions as sets of partially ordered movement constraints which can be expressed in terms of objects and their parts. Constraint-based movement descriptions allow the programmer to assert what is essential for the success of the task while keeping other movement aspects free for optimizing the movement and keeping flexibility. Reasoning may then exploit the modular specification by relating action effects to parts of the movement description, e.g. the pancake will be pushed off if angle between the spatula and the oven is too steep.

In this paper, we suggest a novel constraint-based movement description language and its execution controller which outperforms our predecessor system [7] as it

1) enables the robot to firmly anchor the features employed for motion control in the perceptual apparatus of the system to allow feedback-driven execution and high-level error monitoring.
2) avoids stability issues or kinematic singularities in order to generate higher performance movements.
3) imposes less specification restrictions to fascilitate manual and autonomous generation of motion commands.

To evaluate our system and validate our claims we chose to use the same task as in [7]. We use our low-level system to have a robot push a spatula under a pancake and flip it as depicted in Figure 1. This is a very challenging

* Georg Bartels, and Michael Beetz are with the Institute for Artificial Intelligence and the TZI (Center for Computing Technologies), University of Bremen, Germany. † Ingo Kresse is with Technische Universität München, Germany.

application both from a low-level and high-level point of view. Additionally, it allows us to directly compare our new system to its predecessor [7].

The remainder of this paper has the following structure: After a brief system overview we present our constraint-based motion representation in great detail. Secondly, we introduce the tools we employ for visualization and numeric analysis of constraints. Afterwards, we evaluate the system using the task of pushing a spatula under a pancake, and compare our movement description language to that of [7]. We present validation of our approach by performing the task of flipping a pancake with a real-world robot. Finally, we will conclude the paper with a summary and an outlook on future work.

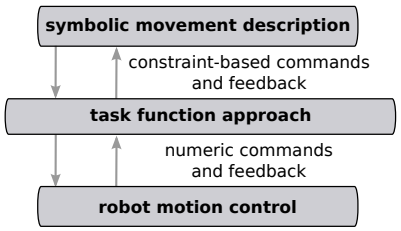


Fig. 2. Conceptual overview of the system: The task function approach maps symbol movement constraints onto numeric control commands and visa versa.

II. System Overview

Consider as an example the pushing of a spatula under a pancake. Informally, this could be described as a movement where (1) *the spatula is always points towards the center of the pancake* (seen from above), (2) *the spatula touches the oven outside of the pancake*, (3) *after the touching the robot pushes the spatula firmly onto the oven while moving towards the center*, and so on.

We propose to model motions such as pancake flipping as partially ordered sets of movement constraints, i.e. motion phases. We believe that a useful movement description language has to exhibit certain desired properties:

- symbolic and qualitative constraints generalize well over different spatial and kinematic setups
- composable constraint sets allow automatic generation
- motion constraints expressed in terms of perceivable object parts allow tracking-based monitoring
- motion specification languages have to cover the richness of state-of-the-art motion control theory

This last requirement asks for a mapping of the constraint-based movement description language into established control engineering frameworks, such as [8], [9], [5], [10]. We believe this is necessary to generate high quality movements which in turn is necessary to successfully achieve sophisticated manipulations such as pancake flipping. We employ the task function approach to perform this mapping from symbol movement constraints down to numeric feedback control of the robot. Figure 2 depicts a conceptual view of the system.

Finally, we provide a short sample movement description in Figure 3. The example corresponds to the informal description of pancake flipping from the beginning of this section. We show sample constraints and one feature definition for the first motion phase, i.e. positioning the spatula front edge over the pancake oven. Each *constraint* relates one *tool feature* on the spatula to one *object-feature* on the oven by using a *feature function*. *Ranges* express the desired goal state of a constraint.

```
features{
  feature{
    name = spatula-front-axis
    ref_frame = /spatula_blade
    type = LINE
    position = [0.0, 0.0, blade_length/2.0]
    direction = [0.0, 0.1, 0.0]}
  ...}
constraints{
  constraint{
    tool_feature = spatula-front-axis
    object_feature = oven-center-point
    function = distance
    range [0.0; oven_radius]}
  constraint{
    tool_feature = spatula-front-axis
    object_feature = oven-plane
    function = height
    range [0.1; +inf]}
  ...}
```

Fig. 3. Sample constraint and feature definitions for the first phase of pancake flipping, positioning the front of the spatula over the oven. The relevant syntactic elements of the language are shown in bold font.

III. Low-level Constraint Representation

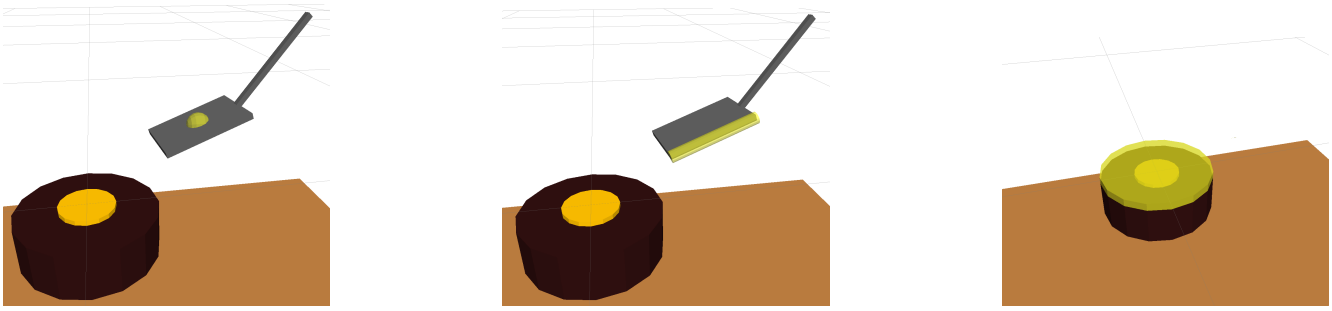
Consider directly translating a high level constraint like *"keep the main axis of the spatula pointed at the center of the oven"* into the control law of a robotic manipulator, while also preserving its semantic information. In this section we present the low-level constraint representation and execution we use to achieve this.

A. Task Function Approach

The control system we propose uses the task function approach [1]. A task function is a differentiable function of the robot's posture. Using it's derivative, the robot is controlled such that the task function yields a desired value. It is also possible to define a task function over different sensor data e.g. the measurement of a force-torque sensor, but the derivative still has to be a function of the robot configuration to enable control.

In particular, we take the approach used in iTaSC [11], in which task functions are defined over the pose of an object that is to be manipulated with respect to a tool which is attached to the robots end-effector. This approach fascilitates generalization to different robots or e.g. two-arm setups in which another robot arm is holding the object.

Consider the poses of the tool $\mathbf{x}_t$ and the object that shall be manipulated $\mathbf{x}_o$. We define the task function scoring



(a) A point feature used to model the center of a spatula.

(b) A line-segment feature which represents the side edge of a spatula.

(c) The oven is modeled use a plane-segment feature.

Fig. 4. The three geometric features supported by the system with example usages: The point, line-segment and plane-segment feature.

the relationship between both objects as $\mathbf{y} = \mathbf{f}_t(\mathbf{x}_t, \mathbf{x}_o)$. Differentiating the task function with respect to time yields:

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}_t}\dot{\mathbf{x}}_t + \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}_o}\dot{\mathbf{x}}_o. \tag{1}$$

Assuming that we can only control the tool and not the object, we simplify (1) to

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{f}_t}{\partial \mathbf{x}_t}\dot{\mathbf{x}}_t = \mathbf{H}\mathbf{t}_t, \tag{2}$$

where $\mathbf{H}$ is called the interaction matrix and $\mathbf{t}_t$ denotes the twist of the tool. As we assume the tool to be rigidly attached to the gripper of the robot we use the robot Jacobian $\mathbf{J}_R$ to derive the control law which relates joint velocities $\dot{\mathbf{q}}$ to the derivative of the feature function:

$$\dot{\mathbf{y}} = \mathbf{H}\mathbf{J}_R\dot{\mathbf{q}}. \tag{3}$$

Calculating the weighted pseudo-inverse of $\mathbf{H}\mathbf{J}_R$ we can derive the necessary joint velocities to obtain the desired changes in the task function values. Please note, we also assumed a good calibration of the tool within the gripper. In previous work we demonstrated how this can be done automatically using visual features of tools like lines, holes or concavities. [12]. Also note, it is necessary to transform the reference points and -frames of both $\mathbf{H}$ and $\mathbf{J}_R$ before combining them and pseudo-inverting the result [13], [14].

In this paper, we propose to construct the task function $\mathbf{f}_t$ by stacking a set of n scalar-valued *feature functions* $f_{fi}(\mathbf{x}_t, \mathbf{x}_o)$:

$$\mathbf{f}_t(\mathbf{x}_t, \mathbf{x}_o) = \begin{bmatrix} f_{f1}(\mathbf{x}_t, \mathbf{x}_o) \\ \vdots \\ f_{fn}(\mathbf{x}_t, \mathbf{x}_o) \end{bmatrix}. \tag{4}$$

Each of the feature functions $f_{fi}(\ldots)$ expresses to which extent a specific spatial relationship between the features of tool and object, such as perpendicularity or distance, holds. Thus, they directly map constraints like *"keep the main axis of the spatula pointed at the center of the oven"* into the control law for the robot.

B. Geometric Features

The basic building blocks of the motion specification framework we present are its *geometric features*. They represent remarkable parts of the tool and object that shall be related to one another. Currently, we have the following geometric features implemented in our system: A point feature, which might represent a small button of an oven, a line feature, e.g. denoting the edge of a tool or a pointing device such as a finger, and a plane feature, that can be used to model planar support surfaces. All of the features have the following properties:

- a reference frame, w.r.t which they are expressed
- an origin vector, denoting the Cartesian position of the feature
- a direction vector, denoting the main orientation of the feature (note that for points this is ignored and for planes this corresponds to the normal of the plane)

Figure 4 shows objects with sample features visualized that are used to model key parts of the interacting objects.

C. Feature Functions

Using the geometric features we define feature functions f_{fi} that map two features onto a scalar value. As already pointed out, feature functions need to be differentiable and depend on the pose of the robot because the tool feature is attached to the end-effector. Also recall that we build the interaction matrix $\mathbf{H}$ by assembling the partial derivatives of the feature functions w.r.t tool motions $\partial\mathbf{x}_t$. The presented system contains – among others – the following feature functions which we use for evaluation:

- *perpendicular*: Equals zero if both feature directions are perpendicular to one another.
- *height*: corresponds to the length of vector between the origins of the two features, projected onto the direction of the first feature.
- *distance*: denotes the length of the vector between the origins of the two features, projected onto the perpendicular of the direction of the first feature.
- *pointing_at*: equals zero if the direction of the first feature is pointing at the second one, i.e. is directed at some point on the line defined by the origin and direction vectors of the second feature.

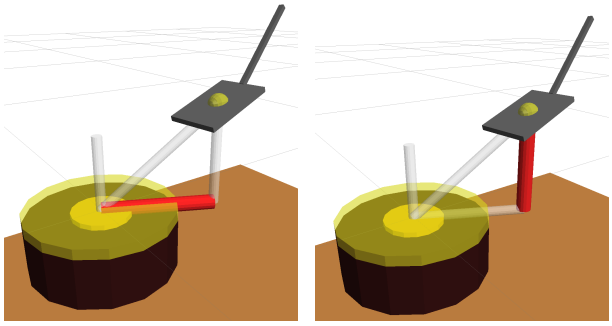


Fig. 5. Visualization of the qualitative meaning of the projected distance (left) and height (right) feature functions between the center of a plane segment and a point.

Using feature functions allows us to express spatial relationships between geometric features as numerical values. For example, we can express that a line in a given situation is pointing more at the center of a plane than in a second setup. Figure 5 shows exemplary visualizations of the height and projected distance feature functions evaluated on a point and plane-segment feature, respectively. Please note, we assume perception to provide the homogeneous transformations between the reference frames of the feature to evaluate the feature functions.

D. Constraints

Finally, we define *constraints* as sets that contain a pair of geometric features, one feature function to describe the intended relationship and a desired value y_d for the feature function. For example, to have a line feature $line_{axis}$ – corresponding to the main axis of our spatula – pointing at the plane $plane_{oven}$ which represents the oven, we just need to specify $y_1 = f_{f1}(\ldots) = pointing\_at(line_{axis}, plane_{oven})$ and $y_d = 0$.

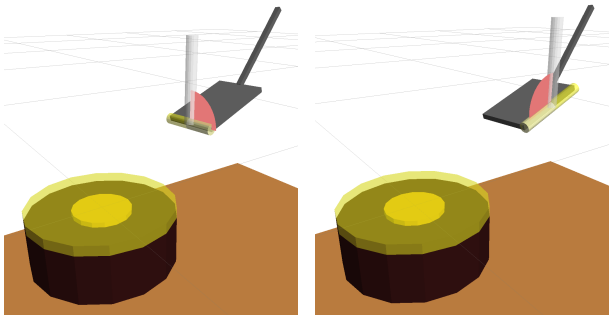


Fig. 6. Aligning a spatula with an oven using two alignment constraints: Restricting the front line-segment of the spatula to be very close to perpendicular to the oven plane's normal causes a good contact (left), while the perpendicular constraint between the side edge and the oven regulates the angle between spatula and oven (right).

A further example is given in Figure 6. Using a set of three constraints we determine the relative orientation of the spatula with respect to the oven. First of all, the front line of the spatula shall be perpendicular to the oven plane's normal, i.e. desired value zero. This constraint causes the front edge of the spatula to form a nice contact with the surface of the oven. Additionally, we set up a perpendicular constraint between the side edge of the spatula and the oven plane which has a non-zero desired value. This value determines the outcome of the pushing action, and learning from demonstration could be a means to obtaining it. Finally, we added a constraint requiring the main axis of the spatula to point at the center of the plane segment representing the oven (not shown in Figure 6). This example highlights how a set of constraints expresses complex spatial relationships in a modular and feature-based way.

E. Controllers for Range-Based Constraints

To incorporate more flexibility into the constraint definitions, we allow for desired ranges $[y_{lo}, y_{hi}]$ instead of just a single desired value y_d for each constraint output. This is useful because symbolic constraints often translate to inequality expressions, e.g. "hold the spatula *over* the pancake". Given the current output values and the desired ranges, the controller calculates the desired velocity $\dot{y}_{des}$ of each constraint.[1] These are then used to solve equation 3 for the desired robot joint velocities $\dot{q}_{des}$ which are sent to the robot hardware.

Furthermore, each 1-D constraint function controller also controls its corresponding weighting factor w influencing the calculation of the weighted pseudo-inverse of $\mathbf{HJ}_R$. Whenever the output value of a constraint function is within the desired range, the controller lowers the weight of that constraint. Reducing the weight of a constraint effectively extends its instantaneous null space, i.e. gives more freedom to other conflicting and unfulfilled constraints.

Since this part of the system is almost identical in design to our previous system we refer the reader to [7] and our code repository for further details.

IV. Analysis of Constraint Sets

A. Visualization of Constraint Sets

Given a set of constraints such as *have a tilted angle between the main spatula axis and the oven plane* and *have no distance between the front edge of the spatula and the oven plane*, we want to visualize the tool motion each of them causes. By visually observing alignment properties or locations of movement axes users may better judge the correctness of constraint sets *during* motion execution.

We present a method to visualize the effect of any task function defined over Cartesian transformations. The input for our method is the interaction matrix $\mathbf{H}$ and the poses (positions $\mathbf{p}$) set in relation by the task function. Therefore our method applies to any task function defined over Cartesian transformations, e.g. the type of task functions we present or virtual kinematic chains. It even generalizes to robot Jacobians.

As shown in Figure 7, we compute and display the instantaneous movements (twists) that affect one and only one constraint at a time. Rotational constraints are displayed

[1]For online computation of interpolation between motion states in constraint space we use the *reflexxes* software library: `http://www.reflexxes.com`

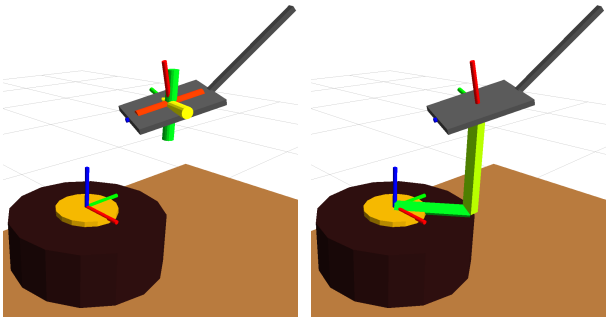


Fig. 7. Visualization of rotational twists (left) and translational twists (right)

by their axis of rotation, translational constraints by their direction.

1) Computing Independent Twists: Given the interaction matrix $\mathbf{H}$, we first compute instantaneous movements (twists) that affect one and only one constraint at a time. By this definition, we call them *mutually orthogonal*.

These twists are more useful for display purposes. In particular, our 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 other constraints, we can infer the location of the axis as a place where rotations around it do not affect other constraints.

Consider the task interaction matrix from equation 2: It computes how a particular twist affects the constraints. Here, we seek the inverse: Given a particular $\dot{\mathbf{y}}$ (one, where only one constraint is non-null), we 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 , i.e. each twist affects only its respective constraint.

2) Visualization of Twists: For visualizing twists, we draw some ideas from screw theory [15] and Plücker line representations [16]. A twist can be interpreted as a rotation around an axis and a simultaneous translation along the same axis (screw movement). For rotational joints, this axis can be visualized: 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, one still needs to calculate the location and length of this line.

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 determines where to show lines.

The reference frame and -point must be known in order to interpret a twist. For visualization, however, we change the reference frame of the twist to the target frame. We refer the reader to [14] for a describtion of how to achieve this.

For the visualization of the instantaneous rotation axis, we 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. 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:

$$\mathbf{p} = \frac{\mathbf{q} \times \mathbf{q}_0}{\|\mathbf{q}\|^2}$$

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

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 $(\mathbf{v}_1, ..., \mathbf{v}_n)$ are given, locations and lengths need to be inferred. This step takes into account the translation $\mathbf{p}$ between tool and object: The translations are required to form a path from the object origin to the tool origin, i.e. they must sum up to $\mathbf{p}$:

$$\mathbf{p} = \sum_{i=n}^{n} \alpha_i \mathbf{v}_i \tag{5}$$

In matrix form this becomes:

$$\mathbf{V}\alpha = \mathbf{p} \tag{6}$$

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 7, right).

We 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 markers.

This visualization tool is well-suited for virtual kinematic chains and other task functions defined over the Cartesian transformations between object and tool. We have used it during system development, design of the pancake flipping application, and to generate the simulation figures in this paper.

B. Combination / Comparison of Constraints

Imagine a situation in which a user of our 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? Similarly, consider the alignments of the spatula's front edge, its side edge and its blade plane (all w.r.t. the oven). How to detect that only two of them are controllable at the same time? Both situations are displayed in Figure 8.

This kind of reasoning is crucial when autonomously combining constraints in the suggested modular fashion. Ideally, we would like the system to only accept constraint sets in which all constraints are independently controllable. Such a situation is given when the rows of the interaction

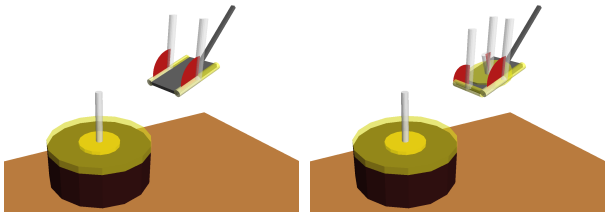


Fig. 8. Dependent alignment constraints (left image: left and right spatula edges, right image: left and front spatula edges and blade plane.m)

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 by the desired ranges of the constraints – we check 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 for any task function if

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

V. Evaluation & Validation

For evaluation we want to compare our abstract low-level movement control machine to its predecessor [7] because it is the most related approach in the literature. The evaluation considers three aspects of the proposed system: Singularities of the underlying representation, execution discontinuities, and the expressiveness of the movement descriptive language. As evaluation task we choose flipping of a pancake with a spatula.

Previously [7], we performed robotic pancake flipping by spanning a 6-DOF VKC between the oven and the spatula (see Fig.1) and controlling its virtual joints. For the first three joints we used cylinder coordinates (angle, distance, and height, a, d, h) which was motivated by the rotational symmetry of the oven. For orienting the tool we chose RPY angles which roughly correspond to the constraints of aligning the front edge (a_f), aligning the side edge (a_s) and pointing direction (p).

We essentially re-create the same chain using feature-based constraints within our framework. As a result, we are able to compare both approaches while performing the same task.

A. Representation Singularities and Discontinuities

To detect a singularity, we can compute the rank of the interaction matrix $\mathbf{H}$ as described in section IV-B. Singularities represent poses in which two constraints become dependent. However, for a controller it is equally important to avoid discontinuities. To detect these, we compute the second

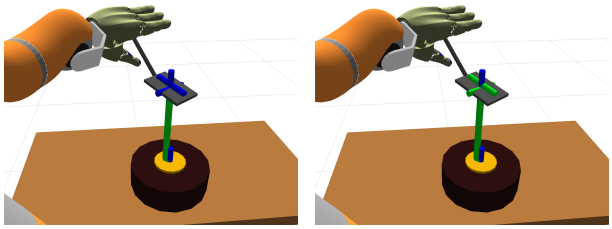


Fig. 10. Constraints at singular positions (left: RPY, right: feature-based). The green rotation/translation axes are fine, the blue axes have a discontinuity and are thus unusable for control

derivatives using three equidistant samples (x_1, x_2, x_3) of the task function $\mathbf{f_t}$ around the current pose:

$$\mathbf{f_t}'' \approx \frac{\mathbf{f_t}(x_3) - 2\mathbf{f_t}(x_2) + \mathbf{f_t}(x_1)}{h^2} \quad (7)$$

Where h is the distance between the samples. If the value of $\mathbf{f_t}''$ exceeds a threshold, we assume to have found a discontinuity.

Using this investigation methodology, we compare the virtual linkage constraints with our 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 constraints for the front-edge and the side-edge of the spatula are semantically independent. Figure 10 visualizes the described tool pose. Near this singularity the following constraints become uncontrollable because of discontinuities:

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

Fig. 11. Comparison of uncontrollable constraints near a singular pose. Marked constraints have become uncontrollable in the given pose. The proposed feature-based formulation retains control over two position constraints which its VKC-based predecessor loses.

For our previous virtual-linkage solution, the angles depend on the cylinder coordinates: At the singularity at 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. This demonstrates that our approach avoids several discontinuities that otherwise require heuristics, e.g. always keeping a minimal distance from the oven center.

Some discontinuities remain, inherent in the problem specification. However, the system can detect them, allowing automatic addition of extra constraints to avoid singularities.

Regarding the angle representations, consider the example given in Figure 12. The spatula is rotated such that both the front and the side edge are at roughly 45 degrees to the oven plane (a_f and a_s), pointing about 20 degrees left to the oven center (a_p).

```
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-main-axis
  object_feature = pancake-center-point
  function = pointing-at
  range [...]}

  ...

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

(left) Movement description language in the predecessor controller [7]: Users need to define the entire fixed-sized 6-DOF VKC. It is noteworthy that constraints which are unnecessary during a motion still need to be modelled, e.g. constraint1. Additionally, the semantics of each constraint strongly depend upon context, and are provided as inputs. For example, constraint1 and constraint6, are both rotations around z. Both, however, carry different semantic meaning.
(right) Our newly suggested description: The number of constraints is unrestricted. Each constraint is defined over its own set of features. Thus, it is possible to have more than two features of interest in a motion description. In our earlier description language this required an additional VKC for every new pair of frames. Finally, the feature functions themselves carry their own semantics. This removes the need for the high-level to connect semantics and functionality.

Fig. 9. Comparing two movement descriptions: On the left, the description as presented in [7] and on the right the newly suggested description.

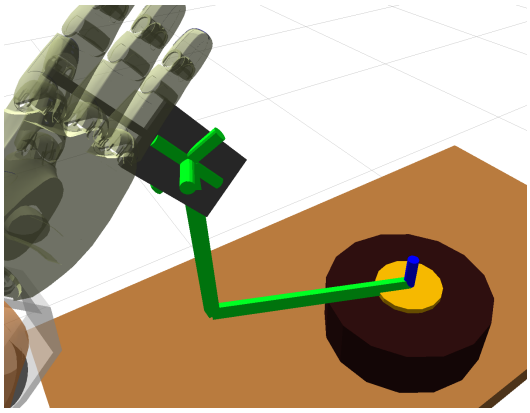


Fig. 12. A non-trivial orientation in which RPY-angles lose their meaning.

Our feature-based constraints correctly report these values as (a_f=-39^o, a_s=50^o, p=18^o). The RPY angle representation looses this meaning: It reports (a_f=-17^o, a_s=-18^o, p=-73^o). While deviations in a_f and a_s might be tolerable in certain situations, the pointing direction has lost meaning completely. At the $(0, 0, 0)$-position, this meaning was still valid.

B. Expressiveness and Usefulness of the Movement Description

To evaluate the expressive differences between our proposed and its predecessor description language we display parts of a corresponding motion description in Figure 9. Both examples describe a sub-motion from a robotic pancake baking task.

Comparing both descriptions, it becomes apparent that our improved language allows for more flexible movement specifications because constraints are allowed in any number and can be ordered arbitrarily. These properties make our representation more modular. Regarding [7], constraints need to be used in groups of six, all part of 6-DOF VKCs – a considerable specification restriction.

In our predecessor system the frames and the virtual joints in the VKCs had to be chosen with great care to ensure chain joints with semantic meaning. It is obviously difficult to automate this reasoning which is a prerequisite for an autonomous high-level system that uses our earlier abstract low-level movement control system. Our newly proposed system, however, encodes semantics in carefully designed feature functions relating object features. As these features are grounded in perception, a possible high-level system no longer needs to decide where to put control frames. Equally, human designers may also find the application programming task simplified.

C. Validation Experiments

We validated our system by performing the task of pancake flipping on a real-world robot. The robot employs both of its arms to complete the assignment, with the movements modelled and executed using the proposed approach. A video

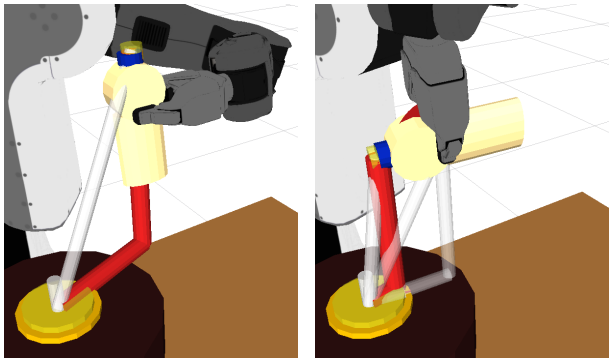


Fig. 13. Visualization of the PR2 pouring content from a bottle onto the pancake oven. The task was modelled with four constraints employing each a different feature function (height, distance, perpendicular, pointing-at). Only three features were necessary to complete the specification: bottle axis, bottle cap, and oven center plane.

recording with detailed setup descriptions can be found in the supplemental material to this paper. Additionally, we modelled the task of pouring a liquid from a bottle (see Figure 13), and are assessing our movement description language on further tasks.

VI. Conclusion

We presented an abstract low-level movement control system. The system is able to perform motions that are specified in terms of sets of constraints over object parts relations, e.g. *move the spatula plane next to the oven plane*, *keep both planes parallel*, while *the main axis of the spatula is pointing at the rim of the pancake*. This interface provides a semantically higher and richer interface, enabling a symbolic high-level system to reason about the effects of actions in terms of the movement parameters that caused them.

Using geometric features, such as points, lines or planes, as control features is a key property of the presented description language. It thus directly grounds the movement specification into the perceptual apparatus of the robot. Additionally, we do not employ an underlying modelling structure like virtual kinematic chains, which greatly improves the versatility and modularity of our system.

We evaluated and validated the system in a robotic pancake making experiment, and compared performance with its predecessor reference system [7]. Simulated and real-world experiments show the applicability of our approach. Additionally, we highlighted the improved modularity and better grounding of our motion representation. In terms of angular singularities and control discontinuities we showed that our proposed system avoids issues that the system presented in [7] encountered. Furthermore, we presented techniques to analyze interaction matrices which simplify debugging.

In the future, we will connect the abstract low-level movement control system to a reasoning system which can exploit the expressiveness of the presented movement description. Consequently, drawing nearer to the goal of equipping robots with the manipulation competence humans exhibit everyday.

Acknowledgments

This work is supported in part by the EU FP7 Projects *RoboHow* (grant number 288533) and *SAPHARI* (grant number 287513). We thank Joris DeSchutter's and Herman Bruyninckx' group from KU Leuven for insightful discussions.

References

[1] C. Samson, M. Le Borgne, and B. Espiau, *Robot Control, the Task Function Approach*. Oxford, England: Clarendon Press, 1991.
[2] L. Morgenstern, "Mid-Sized Axiomatizations of Commonsense Problems: A Case Study in Egg Cracking," *Studia Logica*, vol. 67, no. 3, pp. 333–384, 2001.
[3] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.
[4] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, may 2011, pp. 3828–3834.
[5] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *Int. J. Rob. Res.*, vol. 26, no. 5, pp. 433–455, 2007.
[6] S. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *Robotics, IEEE Transactions on*, vol. 27, no. 5, pp. 943 –957, oct. 2011.
[7] I. Kresse and M. Beetz, "Movement-aware action control – integrating symbolic and control-theoretic action execution," in *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 14–18 2012, pp. 3245–3251.
[8] Albu-Schaffer, A., Haddadin, S., Ott, Ch, Stemmer, A., Wimbock, T., Hirzinger, and G., "The DLR Lightweight Robot – Design and Control Concepts for Robots in Human Environments," *Industrial Robot: An International Journal*, vol. 34, no. 5, pp. 376–385, 2007.
[9] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 1, pp. 43–53, February 1987.
[10] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks," *IEEE Transactions on Robotics*, vol. 25, pp. 670–685, 2009.
[11] R. Smits, T. D. Laet, K. Claes, H. Bruyninckx, and J. D. Schutter, "iTASC: A Tool for Multi-Sensor Integration in Robot Manipulation," in *Multisensor Fusion and Integration for Intelligent Systems*, ser. Lecture Notes in Electrical Engineering, H. K. H. Hahn and S. Lee, Eds. Springer, 2009, vol. 35, pp. 235–254.
[12] I. Kresse, U. Klank, and M. Beetz, "Multimodal autonomous tool analyses and appropriate application," in *11th IEEE-RAS International Conference on Humanoid Robots*, Bled, Slovenia, October, 26–28 2011.
[13] T. De Laet, S. Bellens, R. Smits, E. Aertbeliën, H. Bruyninckx, and J. De Schutter, "Geometric relations between rigid bodies: Semantics for standardization," *IEEE Robotics and Automation Magazine*, 2012.
[14] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-30301-5
[15] M. T. Mason, *Mechanics of Robotic Manipulation*. MIT Press, 2001.
[16] K. Shoemake, "Plücker coordinate tutorial," *Ray Tracing News*, vol. 11, no. 1, 1997.