UNIVERSITY OF BREMEN

BACHELOR THESIS

Towards robotic agents executing underdetermined action description of cutting and pouring

Author:
Vanessa HASSOUNA

Examiner:
Dr. Michael Beetz PhD
Second Examiner:
Dr. René Weller
Supervisor:
Gayane Kazhoyan

A thesis submitted in fulfillment of the requirements for bachelor of science

in the

Institute for Artificial Intelligence

November 12, 2019

Declaration of Authorship

I, Vanessa HASSOUNA, declare that this thesis titled, "Towards robotic agents executing underdetermined action description of cutting and pouring" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Acknowledgements

Thanks to my supervisor, I learned a lot about robotics and was able to find my passion within this process. In addition, I want to thanks my mother, who has always been there for me, supported me at all times and who I trust the most. I would not be here without you two.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The goal is to provide a robot that can solve **daily kitchen activities**. Tasks like grasping, placing or opening drawers are already implemented. However, a task like **cutting and pouring** are missing further actions from the robot. He is able to grasp the knife, but he can't use that tool yet to solve the problem. To work towards that goal this bachelor thesis considers the **missing actions** and tries to implement them as symbolic descriptions. Those should be hierarchically containing motion segments that can be executed in a high level plan.

The goal of this thesis is to enable the robot to perform cutting and pouring actions in a general manner, covering a big number of execution variations. The challenge is that the object to be cut can vary in size, material, position in the world and more.

1.2 Approach

I start with studying the **MPII Cooking Activities Dataset** [2]. The goal was to design a model of the actions as a sequence of motion phases or atomic actions based on how humans perform these tasks.
I made a statistic on how often humans perform an atomic action (e.g. washing or peeling the object). Those I categorized in pretreating, right-hand cutting, left-hand cutting and more (see Chapter 4).

This knowledge led me towards a Theoretical Model. Here I am defining what variations are possible, why I am skipping some special cutting and pouring actions and a rough plan for these actions. Furthermore, I am defining what the atomic motion phases are.
To implement this action I wanted to have a framework that fits my general approach of the plans. The concept of motions and designators of the framework called **CRAM** fits well with the theoretical model. The idea is that a user of this library specifics those underdetermined action and the system automatically execute certain plans based on the object.

1.3 Contribution

The scientific contribution is the design of the **theoretical model**, where the **variations** of the actions are covered and the **generalized** approach is explained. Therefore I developed an approach to specify those actions (cutting and pouring) in a general way. The robot should perform those actions in a general manner, covering a big number of variations. My approach from a low-level to a high-level plan can also be used as a foundation for other implementation of new actions within different frameworks.

1.4 Reader Guide

The structure of this thesis starts with the state of the art 2. Here a few papers are described what their main idea is and what would be interesting for this thesis. Followed by this the foundations are explained, e.g. which framework is used. Next is the Theoretical Model 4 here as previously stated a model is designed for the actions, such that first the implementation details are not important. Then the implementation 5 is explained with all the code that was produced. An overview of how the implemented modules are working together will be given and a rough diagram of how the demo scenario is coded. Followed on this the evaluation 6 part will describe the variations and present the executed code. Finally, a conclusion 7 is made with a summary of the main point of this thesis, as well a discussion about what the limits of the implementation will be held. The future work with the question 'what is next' will be answered in the end.

Chapter 2

State of the art

In the following a few papers are mentioned with their main ideas and contribution. Those papers were read in advance to collect some ideas for this thesis. The sections names are similar to the paper titles.

A Representation for General Pouring Behaviour

Akihiko Yamaguchi and Christopher G. Atkeson introduced a work [10] on pouring with a lot of complex tasks. Their pouring task consist of sub-skills, such as tipping, shaking, squeezing, tapping and flow-control. What they have in common with my approach is that they defined certain preparation processes such as grasping a source container. They also present how many different ways humans have to pour out of the same source container. Their approach is to decomposition the pouring action into sequence of sub skills (e.g. moving an arm $\rightarrow$ $graspingacontainer$ $\rightarrow$ $movingthecontainer \rightarrow flowcontrol$...)
which is similar to my approach. They focused on the flow control which is different from my work, since i focused more on a the way how to generalize two different actions rather then focusing on one.

Cooking for humanoid robot, a task that needs symbolic and geometric reasonings

The paper **Cooking for humanoid robot, a task that needs symbolic and geometric reasonings** [6] contributes to the idea of a good recipe book and breaks down what a recipe database can accomplish for a cooking assistant as a robot. Their recipes can be viewed as an incomplete plan of a high-level task. Furthermore, while acting on objects, the state of the object changes and the database has to be updated, e.g. while cutting, the object will never be the same as before. It can be used differently, as well as needing it to be treated differently.

They implemented a supervisor module that takes actions from the high-level plan and estimates the state of the system. While cutting or afterwards, the robot needs to know where his location is and what action to execute next. With their system, they would not only track but also estimate it.

If something went wrong, the supervisor module asks the task planner for a new plan. They implemented three different methods:

- Failure Backtrack for primitive task

- Change of Symbolic state (door is open 20cm not 18cm leads to motion planner)
- User Constraints (user can always take charge of the robot)

Their primitive tasks always leads to a non-primitive task when it takes action in a plan. For example if the robot should cut something, preconditions needs to be satisfied:

- Where is the cutting tool
- Is my hand free
- Can I grab it

Humanoid Motion Generation System on HRP2-JSK for Daily Life Environment

The paper **Humanoid Motion Generation System on HRP2-JSK for Daily Life Environment** [9] describes a software system for a humanoid robot viewed from a motion generation aspect, by taking helping behaviours as an example of a real-world task using kitchen helping experiments (e.g. Pouring a kettle and serving a tray) which shows that developed three levels of motion generation methods with integration, can be useful for complex real-world experiments. To utilize motion generation modules, it is essential to combine modules that give conditions and constraints.
System design to integrate these modules is the current interests. Throughout the experiment surfaces several technical issues for a humanoid robot in a daily life environment, such as waterproof performance on a washing area and evaluating temperature rise.

Classifying Compliant Manipulation Task for automated Planning in Robotics

The paper **Classifying Compliant Manipulation Task for automated Planning in Robotics** [7] wants to convey how important the environment is for the robot. When a robot is performing some actions, he is traditionally clueless what impact he has on the environment, furthermore why he is doing it. The compliant Manipulation Task figure is an excellent overview to think about in which task the robot has contact with the environment. They are describing some Classification Terms like contact fiction is observed when an object is moved along the surface of another (object in contact).

Cognition-Enabled Autonomous Robot Control for the Realization of Home Chore Task Intelligence

"Robots will constantly be faced with new, unknown tasks they have little knowledge about."[5]

Tum-Rosie making a pancake by using instructions from the worldwide web. The idea is to use all the knowledge from the worldwide web to accomplish a task in the real world is such a rational way to do things. But as well even though the recipes are found and also the knowledge about how long the pancake needs to be on the surface, the robots still need to do all the motion action that takes more effort while planning than to read a recipe.

Chapter 3

Foundations

In the following, the used tools (within this thesis) are explained.

3.1 ROS

Robot Operation System (ROS)[1] is a middleware that provides users to communicate between mutual different frameworks via "ros-msgs". The capability provides defining a value (string or float) as a single message, e.g "String-msgs" field containing a string or "float-msgs" field containing a float. Additionally, wrapping another defined message value where a single message can contain both values (string or float).
Those msgs are send from a client to a server. Which both needs to be implemented in the frameworks that either wants to receive a msgs or send it.

ActionServer, Server, Topics in ros:

There are different types of connections to choose from. The ActionServer is a client/server system which allows the client to cancel their call/requests (containing the msgs) at any point, this benefits for when you require a robot to be interrupted from its movement. As well the server provides feedback about his internal state via defined msgs to the client.
Servers are similar to ActionServers, but the call (goal) cannot be canceled.

The topic requires to be published on so that the subscribers of the topic can receive the call, the advantage of this method is that communication is sent to multiple endpoints. However it does not provide feedback.

3.2 Transform Frame | tf

The tf system [4] in ROS keeps track of multiple coordinate frames and the relationship between them. This is represented as a tree structure, where the frame "map" is the root. The map is understood as the environment of the the world e.g. where the walls are or obstacles in general. It is possible to check all coordinate frame information over the ROS network, this is convenient for when as an example wanting to check "where is the loaf of bread in relation to the robot_base". You're able to transform the position from the bread (which is in the bread coordinate system on x0 y0 z0 and normal orientation) to the robot_base frame which leads for example to bread x1 y1 z0.5 of which is interpreted as the bread is from the robot1 in x-, 1 in y- and 0.5 in z-direction away.

3.3 CRAM

Cram (Cognitive Robot Abstract Machine)[3] is a framework where u can implement, design and deploy Software on autonomous robots. This framework offers multiple libraries for robots, such as geometrical reasoning and fast simulation (called Bulletworld). The Bulletworld [8] allows you to test possible executions in simulation using the same code from a real robot prior to performing in the real world. Testing in a simulation is best practice since testing the solution on a real robot is impractical because of the slow pace of the robot performing certain actions. It also offers tools and interfaces for writing Cognition-Enabled reactive concurrent plans or a Prolog client implementation in lisp which then allows sending prolog queries in JSON format over ROS.

3.4 Designators

Designator containing a chain of key-value pairs of symbols and are developed in CRAM [3]. Three different designator types are implemented and the difference between them is how the system handle those objects. The three supported designators are:

- Object Designator
- Action Designator
- Location Designator

Since the implementation only consist of object and action designator no further explanation for location designator is given.

Object Designator

Object designators are the connection between the CRAM plan and the perception subsystem. The object designator is not resolvable without the perception subsystem which does this task. The properties of the designator are key-value pairs that are used to distinguish the object if this is the object that shall be perceived. For example detecting a knife, which will contain properties of key-value pairs (type,name, ...) will only be resolved successfully when the perception subsystem can see a object with those specific key-value pairs.

Action Designator

Action designators are symbolic action descriptions that can be converted to actual ROS[1] actions goals. We can specify properties of the action designator and we can specify rules how to resolves this action designator. Prolog inferencean alysis if there are facts given, such that all the properties of the action designator are true.

Chapter 4

Theoretical Model

In this chapter the statistic on how often humans perform a atomic action is presented with explanation why certain actions (e.g. pretreating the object) are not further discussed. Followed the Theoretical Model of cutting and pouring is presented with their variations and special cases.

After studying the **MPII Cooking Activities Dataset** [2]. It turned out that humans pretreat the object that they want to cut or pour. Before they do a kitchen activity on an object, they peel or shave the object first. Those pretreat makes it easier for humans to work with it. While examining the **MPII Cooking Activities Dataset** the numer of actions that occured in 70 videos were counted. Table 4.1 shows these statistics. Humans often cut the beginning and end of an object as they do not eat those piece. As previously stated, this is more of something humans do but it's not necessary to perform those action with an object. However, some of these preparations are mostly attributable only to our preferences, such as washing or peeling.
Therefore, since it is not necessary, to pretreat the object, to cut or pour this part is renounced. Furthermore, the focus is on the real cutting and pouring part.

types	frequently
pretreat	24
right hand {cutting}	24
left hand {cutting}	3
pouring ladle {right}	1
pouring ladle {left}	2
pouring cup $\rightarrow$ *bowl*	6
pouring cup $\rightarrow$ *cup*	3
pouring pot $\rightarrow$ *bowl*	2
pouring with spoon	5

TABLE 4.1: Statistics on how often a particular action was performed in the MPII

4.1 Theoretical Model Cutting

There are different variations on how to cut an object. Similarly, there are different objects, each requiring different handling. If humans cut something new, which they have never experienced before, they still find a way to accomplish the action even if in the worst case the object is not quite as handsome anymore.

Exceptional cases of cutting are omitted. These include the cutting of apples, pineapples, oranges or mushrooms and many more, since the motions of the atomic action to cut them is completely different than for other objects. For example an apple you cut in half, you take one half,cut it in half again and repeat. Another limitation is that the PR2 only cut slices and no dice or triangles. Accordingly, the focus is on cutting a loaf of bread, as well as on a sausage with different knives. The table describes selected permutations of the variations. How humans cut a loaf of bread is shown in Figure 4.1.

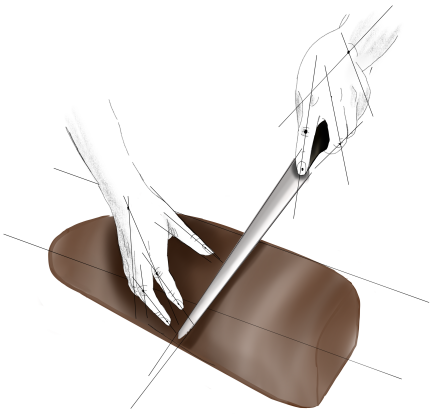

FIGURE 4.1: Human cutting loaf of bread

For example, if the bread is in the left hand (connote with a star) then the bread knife has to be in the right hand. The robot always performs the action with two hands, and the same symbol for the left hand and the right hand corresponds to one combination, i.e. one dual-arm action. In Table 4.2 the possibilities are shown:

hands: / objects:	Left Hand	Right Hand
loaf of bread	⋆	◁
sausage	●	⊙
bread knife	◁	⋆
normal knife	⊙	●

TABLE 4.2: Variations of cutting

The decision with which hand the robot holds the knife should be arbitrary.Humans usually have a more dominant hand (left or right-handed) and therefore mostly try to handle all tasks with this hand. To let the robot do its job as close to human as possible, it is possible to deliberately choose to define the left hand of the robot as

the dominant hand. However, since the robot is capable of doing all the actions and variation as previously described with both hands, there will be no dominant hand given.

The cutting action will start with grasping the tool that is used e.g. a knife followed by multiple slicing motions to cut the object. In the end the tool should be placed down again the timetable for this action is shown in Figure: 4.2.

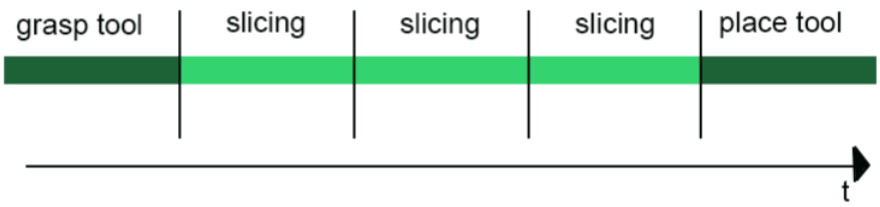

FIGURE 4.2: Timeline of cutting an object

The slicing section consist of 4 atomic actions: approach, lower, lift and retreat as shown in Figure 4.3.

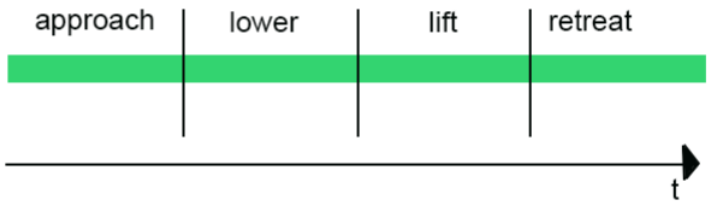

FIGURE 4.3: Timeline for the specific slicing motion

4.2 Theoretical Model Pouring

For the pouring of liquids, humans also have preparatory actions that they perform on the objects, such as washing the glass or mixing one liquid with another. Besides, they often use other objects to pour something out of the object. Capable of using a ladle to get dough out of a bowl or a spoon to get oil out of a bottle without spilling anything makes pouring a more complex task shown as an example in Figure 4.4.

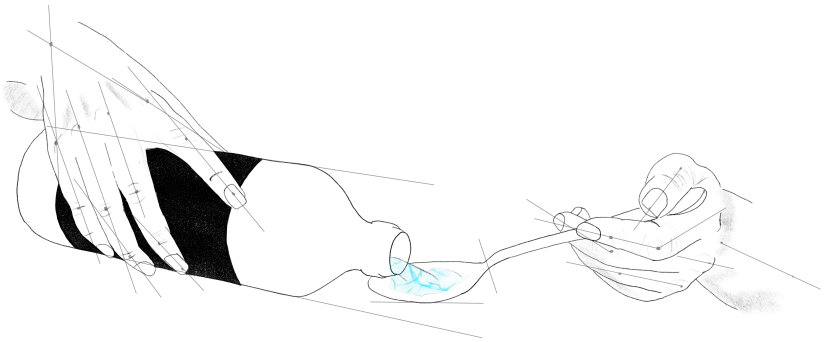

FIGURE 4.4: Human uses spoon for pouring.

Since it is challenging for the robot to touch one object with another, pouring with a spoon is not taking place within this thesis. Additionally, pouring with the ladle would require further movements that have nothing in common with the pouring it self's and accordingly this and the washing is also left out here.

The focus is to pour with each hand. The robot should be able to use different variations of cups and bottles to pour into each other.

Here is an overview of the differences. The Table 4.3 shows which variations of pouring exist. There are two actions hold and pour which the robot can either perform with a bottle, cup1 or cup2. For example if the bottle is hold (represented as $\emptyset$) then it can either be poured into cup1 or cup2 (represented by the same symbol).

actions: / objects:	hold	pour
bottle	$\emptyset$	
cup1	$\odot$	$\emptyset\star$
cup2	$\star$	$\emptyset\odot$

TABLE 4.3: Variations of pouring

The PR2 will approach the object over the second object to pour it. He will start to tilt as much as needed and when the pouring is done he will retract to a given position this is shown as a timeline in Figure 4.5.

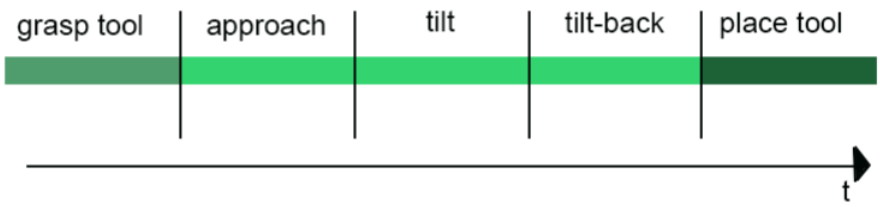

FIGURE 4.5: Timeline for pouring from a object

Chapter 5

Implementation

This chapter discusses the implementation of the modules within the system. Step by step, a more detailed overview will be presented so that subsequent sections can be easily understood. At first, a rough modules overview will be given and by which requirements are constrained - followed by a detailed view of the functionality for each module. Finally, the modules working together will be given with different variation sets to show showcase different variations of executions of the cutting and pouring actions.

5.1 Modules Overview

This section describes the architecture of the software component, implemented in this thesis. Figure 5.1 show case the different modules and their dependencies.

As described in the Theoretical Model, new action for cutting(approach, slice-down, slice-up, retreat) and pouring (approaching, tilt, tilt-back, retreat) are implemented. To achieve further actions **atomic-action-designators** are needed. The plan that will execute the atomic actions, is always the same **move-arm-in-sequence** plan, which gets as input a list of coordinates for the arm to go through and execute given Cartesian arm motions in sequence. Those are symbolic action descriptions that can be converted to actual **ROS** actions goals.

After writing the atomic-action-designators, the action needs to be implemented as a **plan**, this happens in the **cut-and-pour-plans module**. **Low-level failure-handling** is caught here, therefore manipulation failure, such as moving the arm can be handled at this point when the atomic-action-designators are called.

Followed by this, the **cut-and-pour-designators** needs to be defined, those include the new action designators of the new types **pouring** and **cutting**, which is used as an input for the cut-and-pour-plans. The necessary **trajectory** to solve the action is calculated in the trajectory module. To commence the calculation, the object that is acted on must be considered first in the specific objects knowledge module. Here **different objects will lead to another result for the arm kinematics** since their geometrical properties are different from object to object.

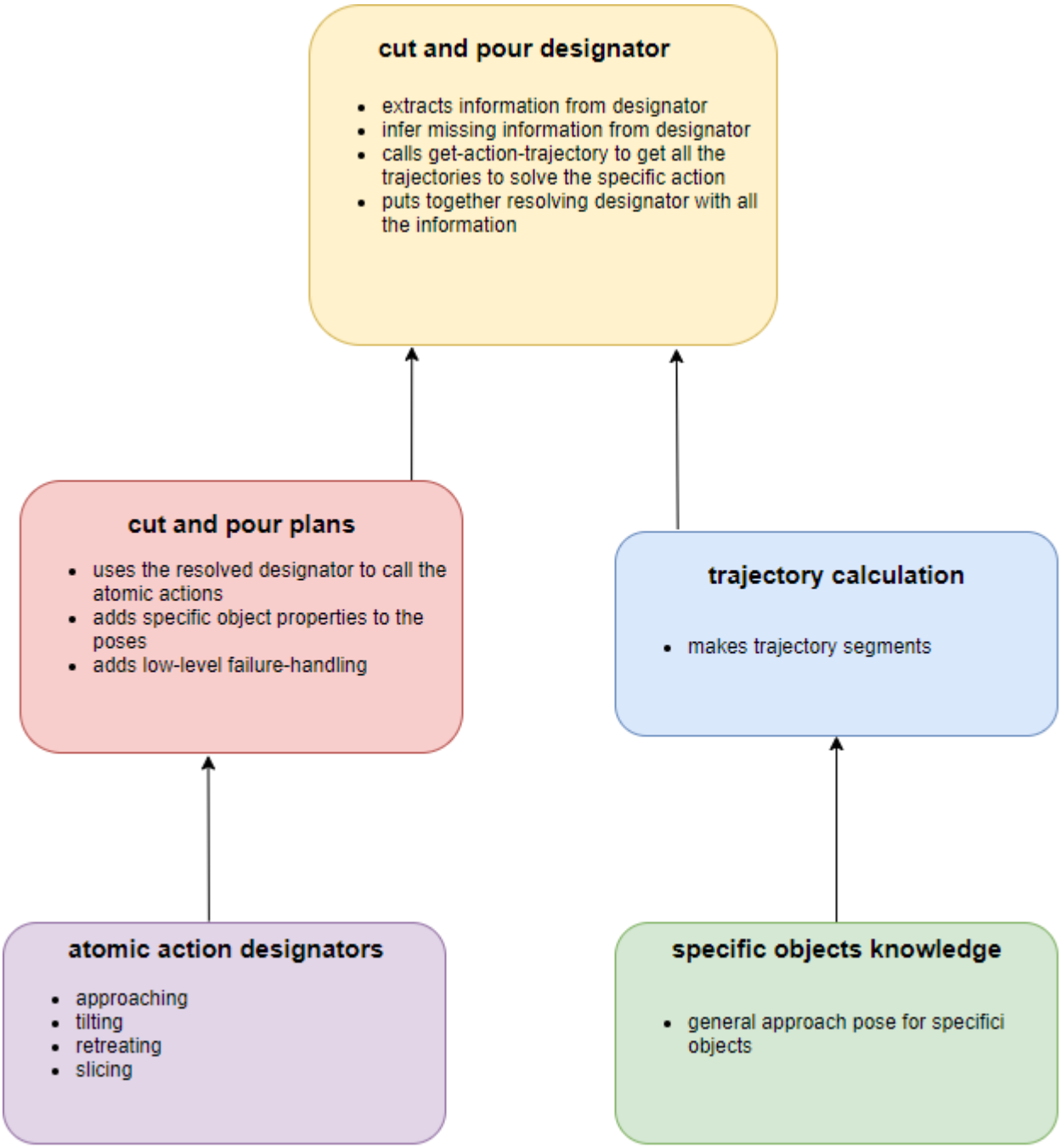

FIGURE 5.1: Modules overview for the implementation of the new actions. Including the atomic action desginators, cut and pour plans, cut and pour designator, trajectory calculation and specific objects knowledge.

In conclusion, the **cut and pour designator** extracts information from the designator, infers missing information and calls **get-action-trajectory** from the trajectory calculation module. All the information are put together in a resolving designator. The **cut and pour plans** are being called with this resolving designator. The given poses (e.g. slicing-up-poses) are then edit to adjust the value depending on the object. The **cut and pour plans** are using the **atomic action designators** to perform the atomic actions.

5.2 Specific Objects Knowledge

The very beginning of the trajectory calculation starts with the object that is approached or acted on in general. For different task, the object needs to be differently manipulated. An example; a bottle has different dimensions as a cup therefore different offsets are needed for an appropriate manipulation-pose (grasp, approach,lift and many more).

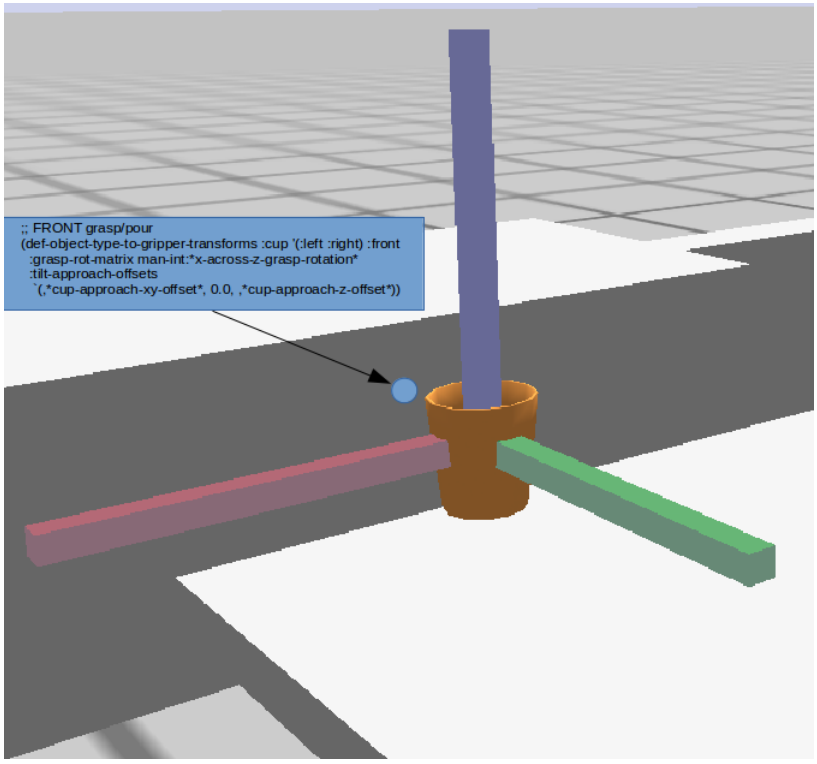

FIGURE 5.2: Cup with pose representation with X-,Y- and Z-Axis and a new defined FRONT pose shown as a blue circle

The middle point of the cup shown in Figure 5.2 is were the X-, Y- and Z-Axis starts. A new definition of a **front-pose** for pouring is defined with the **graps-rot-matrix and tilt-approach-offset**. The **grasp-rot-matrix** contains the information on how the gripper needs to be rotated to be able to reach that pose. The **tilt-approach-offsets** translate this pose further away to be able to pour into the cup. This is now a general pose and can be understood as 'if the edge (usually located on the open side of the object, e.g. bottle-head) that is poured from is at this specific point it is possible to pour with a given angle'. This general pose is changed for each object individually.

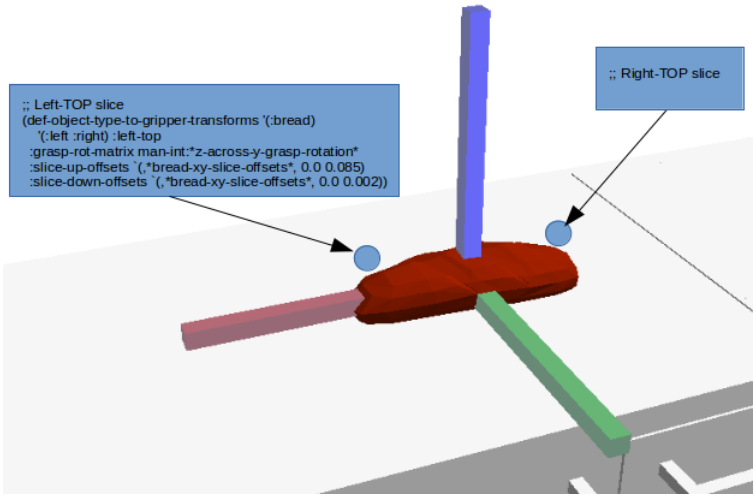

FIGURE 5.3: Bread with pose representation with X-,Y- and Z-Axis and a new defined LEFT-TOP/RIGHT-TOP pose shown as a blue circle

The loaf of bread shown in Figure 5.3 is a new object for the bullet world that was added throughout the process of the implementation. Two new poses were defined to be able to cut the object, and the first pose is with the origin **slice-up-offset** and orientation of **grasp-rot-matrix**. The second pose origin is **slice-down-pose** with the same orientation as the **slice-up-pose**. The same applies to **top-right** and similar to the cup those poses define a general pose were the knife (sharp side) should be to slice the object.

A few more poses that are implemented and follows the same rules as mentioned before:

- bread-left-grasp-pose
- bread-right-grasp-pose
- weissswurst-left-grasp-pose
- weisswurst-right-grasp-pose
- weisswurst-left-slice-up-pose
- weisswurst-right-slice-up-pose
- weisswurst-left-slice-down-pose
- weisswurst-right-slice-down-pose
- cup-back-grasp-pose
- big-knife-front-grasp-pose
- knife-front-grasp-pose

5.3 Trajectory Calculation

This module calculates the trajectories and transforms them into the necessary transform frames. The following section explains the foundation of the modules that were already given in advance.

5.3.1 Foundation

Till this point, as previously described, each object has a general pose on how to act on it. It is necessary to translate this general pose from the transform-frame of the object to the transform-frame of the gripper. With **mapcar** an iteration over each pose is made. Those poses are then translated from the **transform frame of the object** to the **transform frame of the desired gripper**.

5.3.2 Slicing

This function calls the **get-object-type-to-gripper-slice-up-transform** and **get-object-type-to-gripper-slice-down-transform** with given values. This is then made into a trajectory segment, that binds a label-list and a list of the poses with key shown in Listing 5.1.

```
1  (get-action-trajectory ((action-type (eql :slicing))
2
3   (mapcar (lambda (label transforms)
4             (make-traj-segment
5              :label label
6              :poses (mapcar (alexandria:curry #'
   calculate-gripper-pose-in-map bTo arm)
7                             transforms)))
8           '(:slice-up
9             :slice-down)
10          `((,(get-object-type-to-gripper-slice-up-transform
11               object-type object-name arm grasp oTg-std))
12            (,(get-object-type-to-gripper-slice-down-transform
13               object-type object-name arm grasp oTg-std))))))
14
15
```

LISTING 5.1: Get-Action-Trajectory for slicing

5.3.3 Pouring

Similar code to **??** the only difference is the **get-object-type-to-gripper-tilt-approach-transform** which is getting the tilt-approach-pose as shown in Listing 5.2.

```
(get-action-trajectory ((action-type (eql :pouring))

 (mapcar (lambda (label transforms)
           (make-traj-segment
            :label label
            :poses (mapcar (alexandria:curry #'
calculate-gripper-pose-in-map bTo arm)
                           transforms)))
         '(:approach)
         `((,(get-object-type-to-gripper-tilt-approach-transform
              object-type object-name arm grasp oTg-std)))))


```

LISTING 5.2: Get-Action-Trajectory for pouring

5.4 Cut and Pour Designators

In the following the cut and pour designators will be explained. Those are including the new action designators which are containing the trajectories for each task.

5.4.1 Cut Designator

To exectue a cutting action, the robot programmer should create a cuttign designator that look similar to the following:

```
(perform (an action
             (type slicing)
             (object ?object-to-slice)
             (arm ?arm-to-slice)
             (grasp ?slice-position)))
```

The action grounding Prolog rule then transforms this input action designator into the following output designator:

```
(designator :action ((:type :pouring)
                 (:object ?current-object-desig)
                 (:object-type ?object-type)
                 (:object-name  ?object-name)
                 (:arm ?arm)
                 (:grasp ?grasp)
                 (:left-approach-poses ?left-approach-poses)
                 (:right-approach-poses ?right-approach-poses)
                 (:left-tilt-poses ?left-tilt-poses)
                 (:right-tilt-poses ?right-tilt-poses))
                       ?resolved-action-designator))
```

The code shown in Listing 5.3 is very similar to the previously mentioned atomic-action-designator 5.6 and other action designators such as picking-up. Since all the information is extracted here from the object designator and even missing information are inferred, for example how the object is turned towards the robot, how much effort is needed for the object and how far should the gripper be open to doing certain actions. The code-base was already given and is not a new implementation but just used differently.

```
(<- (action-grounding ?action-designator
                              (cut ?resolved-action-designator))
      (property ?action-designator (:type :slicing))

      ;; extract info from ?action-designator
      (property ?action-designator (:object ?object-designator))
      (current-designator ?object-designator
                                 ?current-object-desig)
      (property ?current-object-desig (:type ?object-type))
      (property ?current-object-desig (:name ?object-name))

      (-> (property ?action-designator (:arm ?arm))
          (true)
          (robot-free-hand ?_ ?arm))
      (get-object-transform ?current-object-desig
                            ?object-transform)

      ;; infer missing information like ?grasp type, gripping
```

```
19      ;; ?maximum-effort, manipulation poses
20      (calculate-object-faces ?object-transform
21                              (?facing-robot-face
22                                ?bottom-face))
23
24      (property ?action-designator (:grasp ?grasp))
25
26      (get-action-gripping-effort ?object-type
27                                  ?effort)
28      (get-action-gripper-opening ?object-type
29                                  ?gripper-opening)
30
```

LISTING 5.3: Cut Designator extracting info

More interestingly is how the trajectories are calculated and inserted to the action designator which is shown in Listing 5.4.

```
1  (-> (equal ?arm :left)
2          (and (get-action-trajectory
3                                :slicing ?arm
4                                         ?grasp
5                                         ?objects
6                                         ?left-slicing-pose)
7               (get-traj-poses-by-label
8                                  ?left-slicing-pose :slice-up
9                                  ?left-slice-up-poses)
10              (get-traj-poses-by-label
11                                 ?left-slicing-pose :slice-down
12                                 ?left-slice-down-poses))
13
14    ;;put together resulting action designator
15    (designator :action [...]
16                                    ?resolved-action-designator))
17
```

LISTING 5.4: Cut Designator Calculating Trajectory

If the given arm is equal to the key :left, then get the action-trajectory for :slicing with: given arm, grasp the object. Those trajectories are then saved inside the variable **?left-slicing-pose**. A trajectory contains two lists, the first contains labels and the second the poses transformed to "map". With **get-traj-poses-by-label**, it is possible to get the corresponding trajectory to the key, for example :slice-up. The same applies to the right side. At the very end, the resulting action designator is made with all the information that was extracted or calculated earlier.

5.4.2 Pour Designator

For pouring the inferring missing and extracting information is equally to the cut-action-designator, therefore its skipped here but exist in the original code. For the pour trajectory is calculated and saved as **?left/right-pouring-pose** as shown in Listing 5.5. The resulting **approach-poses** are then given as an in input for **get-action-trajectory** type :tilting to calculate the **?left/right-tilt-poses** corresponding to where the gripper already is.

```
;; calculate trajectory
   (equal ?objects (?current-object-desig))
   (-> (equal ?arm :left)
       (and (get-action-trajectory
                                    :pouring ?arm
                                             ?grasp
                                             ?objects
                                             ?left-pouring-pose)
            (get-traj-poses-by-label
                                     ?left-pouring-pose :approach
                                     ?left-approach-poses)
            (get-action-trajectory
                                   :tilting ?arm
                                            ?grasp
                                            ?objects
                                            :tilt-approach-poses
                                               ?left-approach-poses
                                            ?left-tilt-poses))

 ;;put together resulting action designator
 (designator :action [...]
                                  ?resolved-action-designator))

```

LISTING 5.5: Pour Designator

5.5 Cut and Pour Plans

The atomic-action-designator will be used within the plan of cut and pour. Before implementing those plans, the requirements for each action should be precise. In the following, the plan for cutting and pouring are explained.

5.5.1 Cut Plan

In the following the detailed explanation of the cut plan is given. The input of the function is shown in Listing 5.6

```
(defun cut (&key
              ((:object ?object-designator))
              ((:object-name  ?object-name))
              ((:arm ?arm))
              ((:left-slice-up-poses ?left-slice-up-poses))
              ((:right-slice-up-poses ?right-slice-up-poses))
              ((:left-slice-down-poses ?left-slice-down-poses))
              ((:right-slice-down-poses ?right-slice-down-poses))
            &allow-other-keys)
  [...]

```

LISTING 5.6: In cut and pour plans the function definition for cut

The required keys for the function **cut** explained:

- The **?object-designator/?object-name** variables are used for calculating an axis aligned bounding box for the object, such that it can be worked with these dimensions, to provide actions that are concerning those properties.

  The robot will hold the object with one hand on the very edge of the object. The reason therefor is that during the cutting process, the object gets smaller and smaller. For the safety of the robot, he will hold the object the whole time at one spot.

  To determine how often the robot should slice an object the approach is to use the 'axis aligned bounding box (AABB)' given from the object information to determine how often the robot should slice the object and how big the steps are between each slice. Following that approach, the slices amount and slice thickness will depend on the object size.

  Cutting the loaf of bread will be performed with a bigger and sharper knife. Humans movement to cut bread is similar to a small sawing motion. However, to simplify the model, only chopping motions are considered. The robot will approach the object with the knife, then he moves his hand down with some pressure so that the bread is cut through. If one slice is cut, he will raise the knife again and approaches the next position to slice.
- For the actual movement of the arm the **?left/right-slice-up/down-poses** are given with **?arm (:left, right)**. The **slice-up-pose** is the approach pose for the object and the slice down is the same pose just depending on the object height it is localized further down. The combination of those two makes the motion of slicing.

To calculate how often objects should be cut, specified on their parameters, the following equations were used, with **N = how often to slice**, **L = thickness of one slice** and **G = gripper position offset while holding the knife**:

$$N = (2 * aabb(object).x)/d$$

$$L = (2 * aabb(object).x)/N$$

$$G = aabb(object).y)/2$$

For **N** the x value of the axis aligned bounding box is taken and multiplied by 2, since the framework is only calculating the **aabb** (already implemented function in the framework) for the half of the object, depending on where the origin is. Following the value is divided by **d**. The value of **d** is currently empirically estimated as 0.02. The **L** takes again the x from the aabb and devides that by N.

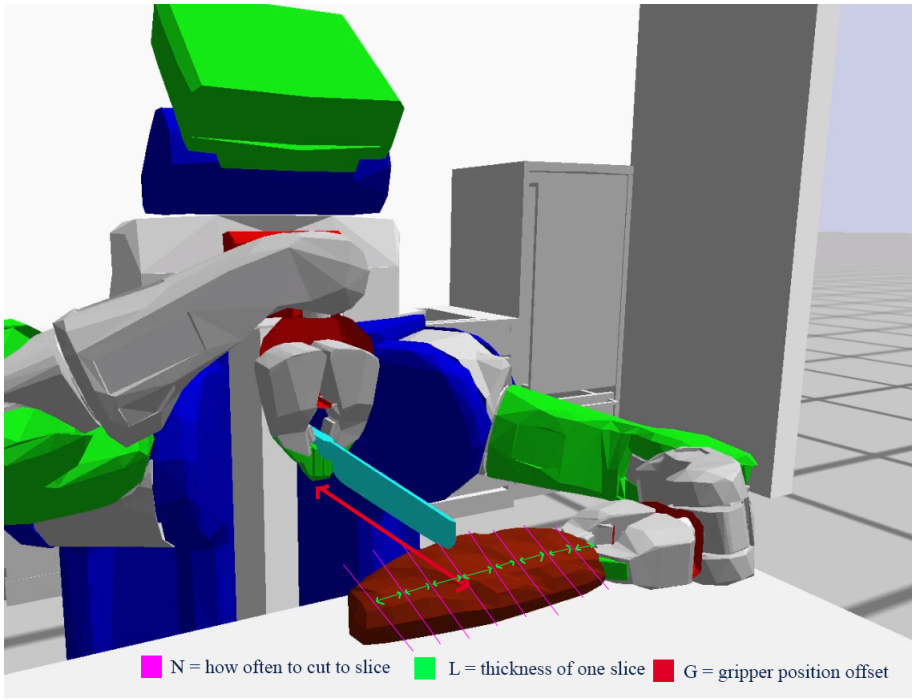

FIGURE 5.4: PR2 cutting the loaf of bread with annotated equation.

Since the robot is not cutting with his bare gripper but with an appropriate object, the value **G** is there to decide how far away the gripper should be from the object shown in Figure 5.4. These calculations depend on the y-dimension of the aabb from the object divided by 2.

In the following Listing 5.7, the first slice is planned. First, the object is approached, then the knife will slice downwards and finally retreated to the approach pose called slice-up.

```
      ;;approaching
      (perform
       (an action
                 (type approaching)
                 (left-poses ?left-slice-up-poses)
                 (right-poses ?right-slice-up-poses))))
      ;;slicing
      (perform
       (an action
                 (type slicing)
                 (left-poses ?left-slice-down-poses)
                 (right-poses ?right-slice-down-poses))))
      ;;approaching
      (perform
       (an action
                 (type approaching)
                 (left-poses ?left-slice-up-poses)
                 (right-poses ?right-slice-up-poses))))

```

LISTING 5.7: In cut and pour plans the function cut performing the actions

In the following Listing 5.8 the main loop for the slice action is shown.

```
    ;; here is the slicing loop, loop as often as we want to make 1
    slice
    ;; note: 3 slices are left out to save the robot from slicing his
    gripper of
    (dotimes (n (- n-times-slice-value 3))
      (let ((?left-slice-up-adjustment-poses
              (mapcar (lambda (slice-left)
                        (translate-pose slice-left
                            :y-offset (- length-one-slice)))
                      ?left-slice-up-poses))
                [...]
          ;;approching / slice up
          (perform
           (an action
                     (type approaching)
                     (left-poses ?left-slice-up-adjustment-poses)
                     (right-poses ?right-slice-up-adjustment-poses))))
          ;;slice down
          (perform
           (an action
                     (type approaching)
                     (left-poses ?left-slice-down-adjustment-poses)
                     (right-poses ?right-slice-down-adjustment-poses)))
[...]

```

LISTING 5.8: In cut and pour plans the function cut looping actions

Dotimes is the main loop in cutting, which means that one iteration is equivalent to one slice. The loop is called as often as the robot should slice, as previously described and calculated. A restriction is that three slices are left out for the safety of the robot. The original slice-up and slice-down poses are translated into a new pose. This new pose is equivalent to the **x-axis** and **z-axis** of the gripper but moved on the **y-axis**. This means that the robot-arm moves left or right, depending on the arm,

over the object. Those adjusted approach poses are now more to the left or right of the object. The function **translate-pose** is a helper function that transforms the slice poses from "map" to "base_footprint", applies the offset to the poses and transforms the adjusted poses back to "map". The transformation itself is used from a library in CRAM [3] which implement the use of ROS tf [4].The loop continues with the translated approach-, slicing-down- and slicing-up-poses.

5.5.2 Pour Plan

In the following the detailed explanation of the pour plan is given. The input of the function is shown in Listing 5.9.

```
(defun pour (&key
               ((:arm ?arm))
               ((:grasp ?grasp))
               ((:left-approach-poses ?left-approach-poses))
               ((:right-approach-poses ?right-approach-poses))
               ((:left-tilt-poses ?left-tilt-poses))
               ((:right-tilt-poses ?right-tilt-poses))
             &allow-other-keys)
  [...]

```

LISTING 5.9: In cut and pour plans the function definition for pour

The required keys for the function **pour** explained:

- The **?arm** and **?grasp** keys are used to find out what object is in the given arm and how far the gripper should be away from the object, that is poured into, to not collide with the object inside the gripper.
- The **left/right-approach-poses** and **left/right-tilt-poses** are poses that describes where the gripper has to be to perform the action.

The object that the robot is pouring from is essential if its to small it needs to be more near to the object that is poured into, if it's higher then should be further away. Therefore, taking this into account the given poses will be translated into adjusted poses, those are containing the offsets. Those are calculated with:

$$Y = aabb(object).y)/2$$

$$Z = aabb(object).z)/5$$

A similar calculation is performed, as previously mentioned, with an aligned bounding box.

```
      (an action
                (type approaching)
                (left-poses ?left-approach-poses)
                (right-poses ?right-approach-poses))))

    (perform
     (an action
                (type tilting)
                (left-poses ?left-tilt-poses)
                (right-poses ?right-tilt-poses))))

    (perform
     (an action
                (type approaching)
                (left-poses ?left-approach-poses)
                (right-poses ?right-approach-poses))))))

```

LISTING 5.10: In cut and pour plans the function pour performing the actions

After the poses are translated the action can be called shown in Listing 5.10. Those are done similar to slicing the object. First, the robot will approach the object with either **?left-approach-poses** or **?right-approach-poses**, then the gripper will tilt 100 degrees and then tilt back to the first pose. The robot will always tilt towards the object since its taken into account how the robot approaches the object.

5.6 Atomic Action Designators

In this following module, Prolog inference **analysis** if there are facts given, such that all the elements of the body are **true**. In that case, the variables are used to evaluate the head. The action-grounding binds a tuple of commands and specific action parameters. The code is shown in Listing 5.11.

```
(def-fact-group cut-and-pour-atomic-actions (action-grounding)

  (<- (action-grounding ?action-designator (move-arms-in-sequence
                                            ?
    resolved-action-designator))
     (or (property ?action-designator (:type :approaching))
         (property ?action-designator (:type :tilting))
         (property ?action-designator (:type :tilting-back))
         (property ?action-designator (:type :retracting))
         (property ?action-designator (:type :slicing)))
     (property ?action-designator (:type ?action-type))
     (once (or (property ?action-designator (:left-poses ?left-poses))
               (equal ?left-poses nil)))
     (once (or (property ?action-designator (:right-poses ?right-poses))
               (equal ?right-poses nil)))
     (designator :action ((:type ?action-type)
                          (:left-poses ?left-poses)
                          (:right-poses ?right-poses)
                          (:collision-mode :allow-all))
                 ?resolved-action-designator)))

```

LISTING 5.11: In cut and pour plans the function pour performing the actions

The rules that are specified here in the body are interpreted as the following: 'does the action-grounding contain a key-value pair that is of (:type :approaching), or is there one with (:type :tilting)?'. The same rules follow for the whole body, e.g. 'is either a left- or right-poses given?'. After this, the resulting action designator is put together.

If and only if all body elements are true, the head will be evaluated. Here the move-arms-in-sequence function that instantiates the movement is being called with ?resolved-action-designator. Please remind the variables with '?' in the front are prolog-variables.

Chapter 6

Evaluation

This chapter evaluates the codebase. Therefore a demo-scenario was created to show how generic the code is when tested with different setups. In the following, all possible objects are spawned and acted on from two different sides of the kitchen.

6.1 Setup cutting a loaf of bread | Kitchen island

The plan for cutting

Before:

1. One big knife is at the kitchen island or sink area
2. The object is located on the surface of the kitchen area

The Action of cutting:

1. Detect the knife that is appropriate for the object shown in Figure 6.1
2. Try to grab the knife
3. Drive back to park position
4. The gripper that is free holds the determined object shown in Figure 6.1
5. The gripper with the knife approaches the object (depending on the arm either left or right side of the object)
6. Move knife down shown in Figure 6.2
7. Move knife up
8. Approach next slice position with the knife and repeat step 7,8 and 10

The robot stops cutting as soon as all the previous calculated slices have been made.

Shown in Figure 6.1 the loaf of bread is on the kitchen island, and the robot is picking-up the knife with his right hand.

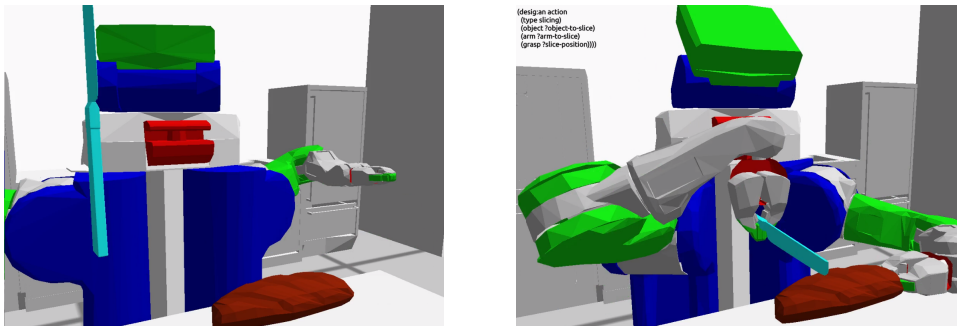

FIGURE 6.1: Loaf of bread and big knife spawned on the kitchen island. On the left figure the PR2 detects the knife and on the right he is performing the hold action with his left-arm.

The loaf of bread was hold with the left arm on the very edge of the object. The right hand is performing the cutting motion shown in 6.2. The last slicing pose for the right hand. As previously mentioned a safe space is given, such that the PR2 does not hurt himself.

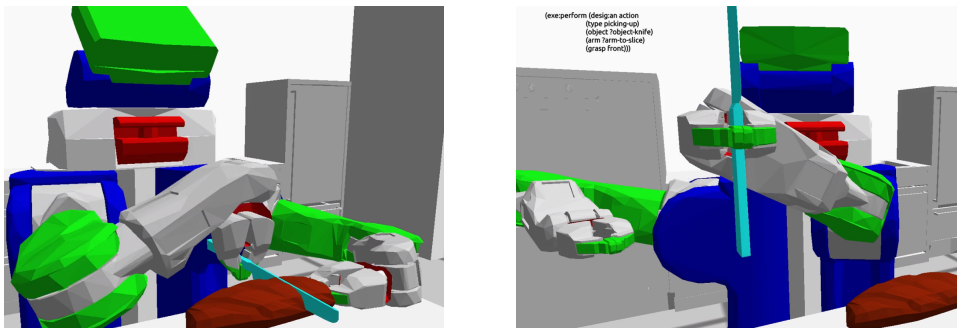

FIGURE 6.2: Loaf of bread and big knife spawned on the kitchen island the left figure shows how the PR2 is performing the last slice pose with the right-arm and on the right he performs a pick-up action on the knife with the left-arm.

The same action is called started from 6.1, except that this time the left arm picks up the knife. The calculation for the approach and slice poses are working for each side of the bread showed in 6.3. Note: save space is given for this side as well.

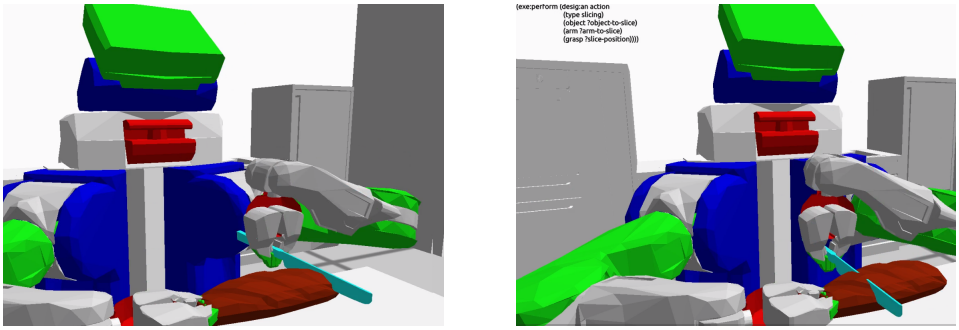

FIGURE 6.3: Loaf of bread and big knife spawned on the kitchen island. The PR2 is holding the bread with his right-arm shown on the left figure and performs the last slice on the right figure.

6.2 Setup cutting a loaf of bread | kitchen sink area

The location of all the objects changed utterly. The PR2 turned 180degrees, and the Objects are now located at the **kitchen sink area**. That means they're having different positions from the setup on the kitchen island area. The big knife is detected and picked up with the left arm of the robot (notice that the sharp edge of the knife is still "forward") as shown in Figure 6.4.

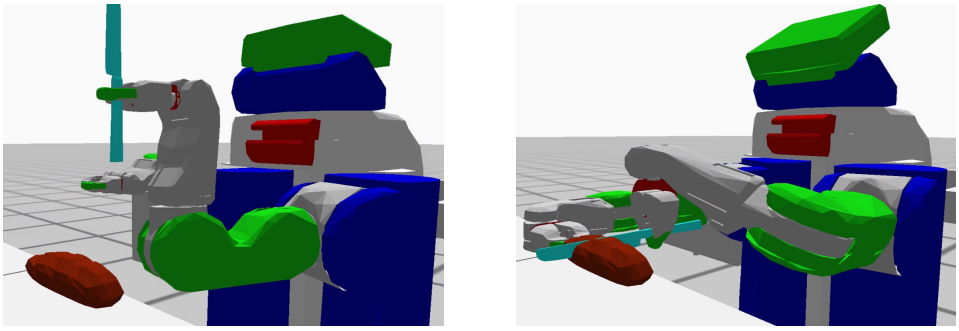

FIGURE 6.4: Loaf of bread on the kitchen sink side hold with left- and sliced with right-arm

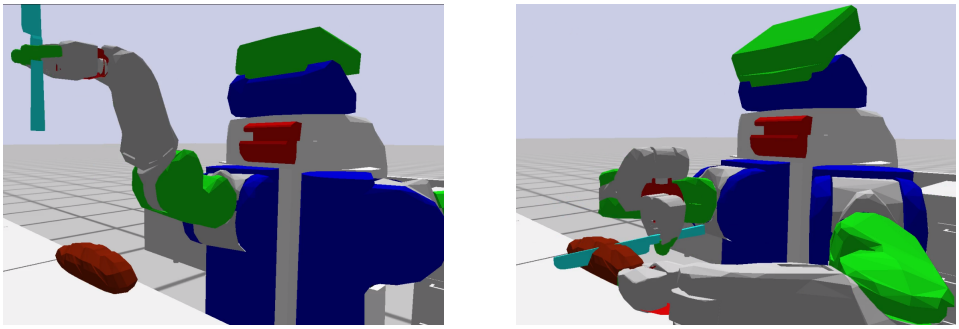

FIGURE 6.5: Loaf of Bread on the kitchen sink side hold with right and sliced with left arm

The arm changed from left to right arm to pick-up the knife and slice the bread in Figure 6.5.

6.3 Setup cutting a weisstwurst | Kitchen island

In this setup the knife and the position of the knife changed, but the PR2 is still able to pick it up shown in Figure 6.6. Furthermore the bread changed to a weisswurst which also leads to a new position. The weisswurst is located more to the middle of the island.

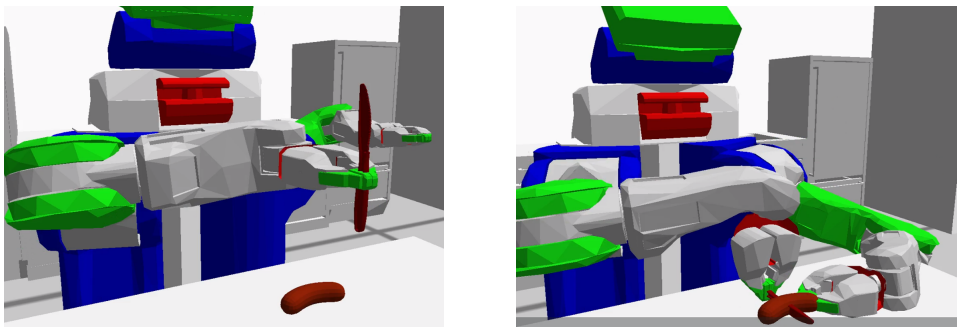

FIGURE 6.6: Weisswurst on the kitchen island side. Knife picked-up with right-arm to perform slicing action. The weisswurst is hold with left- and sliced with right-arm

However, not only the last slice is significant also where the first slice is made (it should not be in the air). As shown on 6.7 the weisswurst is sliced the first time correctly.

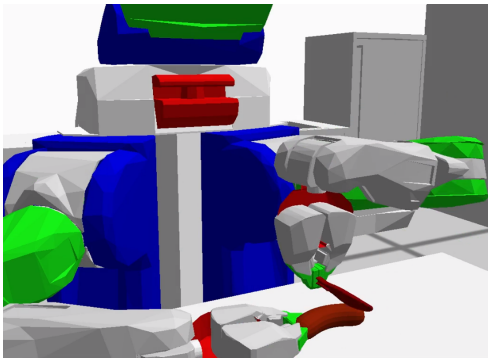

FIGURE 6.7: Weisswurst first slice pose with left- and hold with right-arm

6.4 Setup cutting a weisswurst | Kitchen sink area

As in the setup for the kitchen island area, again the location of the **weisswurst** and **knife** changed compared to the loaf of bread, and the big knife shown in Figure 6.8, yet the robot is executing the same code.

Cutting the weisswurst with the left arm is as well in this location possible as shown in Figure 6.9.

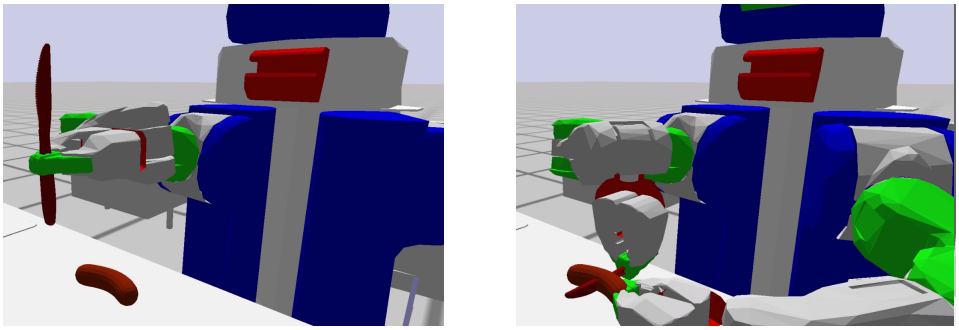

FIGURE 6.8: Weisswurst on the kitchen sink side. Knife picked-up with right-arm to perform slicing action. The weisswurst is hold with left- and sliced with right-arm

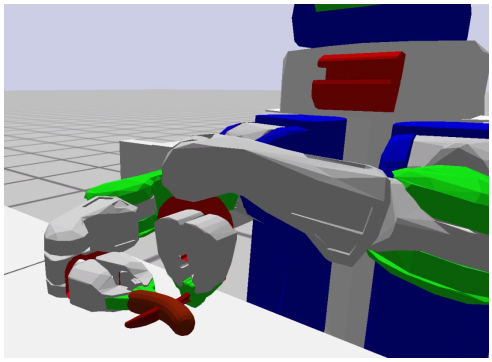

FIGURE 6.9: Weisswurst slice pose with left and hold with right arm

6.5 Setup pouring | Kitchen island

The plan for pouring

Before:

1. A cup to pick-p
2. A bottle to pick-up

The Action of pouring:

1. Pick-up the two objects
2. Drive back to park position
3. Determine where the object to pour into is
4. Approach object with one hand shown in Figure 6.10 and Figure 6.12.
5. Tilt the object inside the hand shown in Figure 6.10 and Figure 6.12.
6. Tilt back
7. Retreat
8. Use different arm shown in Figure 6.11 and Figure 6.13.

The robot stops pouring as soon as all the previous calculated time has passed by.

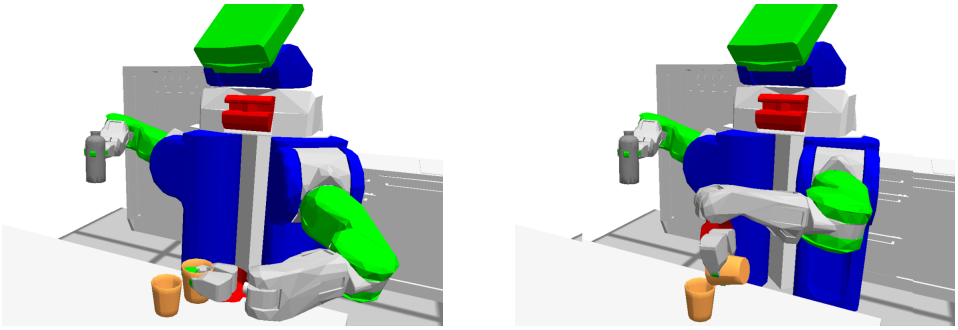

FIGURE 6.10: Kitchen island | On the left figure the cup2 is approached with the left-arm and on the right figure from the source object cup1 is poured into the target container cup2.

The left-arm already picked up a cup, now approaching the second cup to pour into from the left side (robot view).

After approaching the cup tilt 100degrees to pour into the cup.

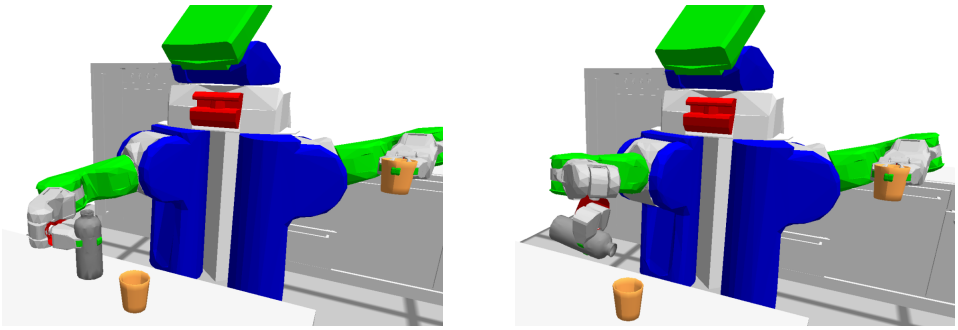

FIGURE 6.11: Kitchen island | On the left figure the cup2 is approached with the left-arm and on the right figure from the source object bottle is poured into the target container cup2.

The right arm already picked up a bottle, now approaching the standing cup2 to pour into from the right side (robot view). The cup1 will be tilted 100degrees to the left side (view from the robot) to pour into the cup.

6.6 Setup pouring | Kitchen sink area

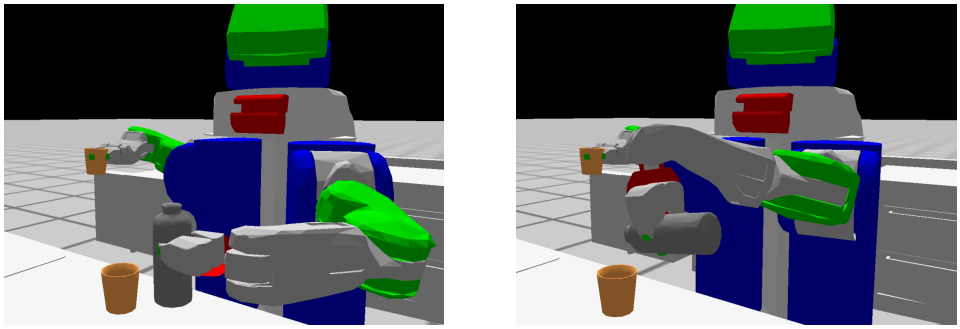

FIGURE 6.12: Kitchen sink | Pouring from the source object bottle into the target object cup2.

As described in 6.2, the position and rotation of the cups change in this setup, and are now located in different locations.
Tilting the left arm to pour from the bottle into the cup.

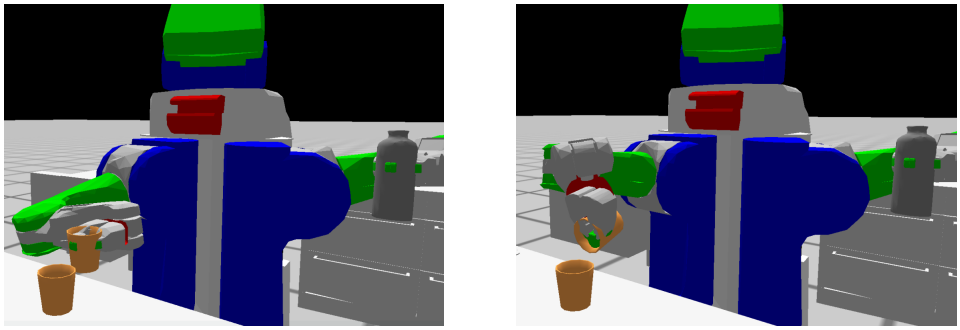

FIGURE 6.13: Kitchen sink | Pouring from the source object cup1 into the target object cup2.

Furthermore, it's not only possible to approach and pour from the left- or right-side, but also from the back- and front-side. In the Figure 6.13 the right arm approaches the cup from the back side and tilts 100degrees forward to pour into the cup.

6.7 Analysis

The different setups were all called with the same designators for slicing and pouring. Therefore also the calculation of the slice poses and the pouring poses are all **not** dependant on the position of the object in the world. This knowledge can now be used to make a more dynamic demo scenario and to solve daily kitchen activities.

Chapter 7

Conclusion

The following summarizes what the thesis contributes and and what inferiorities it may be subject to.

7.1 Summary

Within this work, new general actions were implemented into **CRAM**[3]. The actions are one-handed pouring and cutting.

To make a cutting action, a general pose was introduced for two objects, which is then changed based on the size of the object. That means that for each object, **only** two new Poses are necessary to be able to cut this object.
If the implementation of all those poses were different in general and not based on a calculation onto those two poses, then this would lead to at least **2*N** new poses (with n being the number of slices). Since the general poses are in relationship to the object, it does not matter where the object is in the world, because all the calculation still working the same.

For pouring only one new pose was added: the **tilt-approach-pose**. This was implemented for back, front, left-side and right-side of the cup. When adding this poses to the bowl or a pot, pouring will work on those too. The object from that is poured from only matters for the calculation of the end-pose since a bottle is longer than a cup.

7.2 Discussion

A limitation is that the orientation of the object can not be different since left- and right-side are defined. If that changes the robot would try to grab with his left arm all the way to the wrong side and the other way around (even the PR2 with all of his joints is not able to do so). However, since this problem was clear from the beginning the chosen orientation was not random. If these actions are tested on the real robot the perception-system (that is used in the institute that supervised the thesis) will deliver a pose for a detected object that is similar to the spawn orientation in the bullet world (x-axis is always the long side of the object, z-axis as height and y-axis is away from the robot). **Another limitation** is that only those two objects are mutable at this point, but as previously stated it only takes two new poses to define the cut

for a new object and that is done in no time.

7.3 Future Work

The future work with these new actions is fascinating. The first step would be to test if these new actions works on the other robots in the bullet world (which they should). At least for **BOXY** who has 2 arms as the PR2 and a lot of freedom in degrees in her arms, she should be able to execute those actions.

Next would be to add more objects that the actions can be used on. The bread and the big-knife were inserted within this bachelor thesis into the bullet world to even have enough variations to prove the generic property of the code. Adding a bottle to pour into should be very easy, since it will act as a cup. More interesting is to add objects that will cover the special cases that were mentioned in the Theoretical Model 4. It is also possible to concentrate on one action and specify a more complex task similar to Yamaguchi and Atkeson approach [10], since various of size is already covered by the objects that were used in this bachelor thesis, various of shape would be the next to focus on e.g. cutting an apple. When it comes to pouring a two-hand variation would be pleasing as parts of it are already been developed through the thesis. This could be done by pouring out of a pot shown in the Figure below.

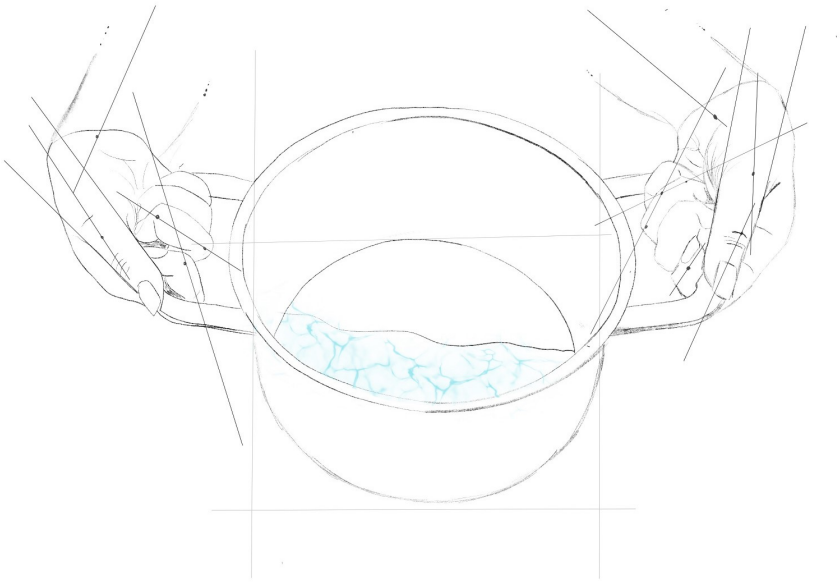

FIGURE 7.1: Human pouring out of a pot.

After all this, the code should be tested on the real robot, which is possible since the code was developed in CRAM [3]. This gives us the functionalty to execute the same code on the real robot as in the bullet world. Here the challenge is that the robot has to work with an external perception system, which was already considered before the start of the implementation. Furthermore, the robot should first try to only cut or pour with 'air' and not have a knife in his hand or liquid in the container.

Bibliography

[1] Powering the world's robots. https://www.ros.org/. [Online; accessed 12/11/2019].

[2] Mpii cooking activities dataset. https://www.mpi-inf.mpg.de/departments/computer-vision-and-machine-learning/research/human-activity-recognition/mpii-cooking-activities-dataset/, 2014. [Online; accessed 12/11/2019].

[3] CRAM: Cognitive Robot Abstract Machine. http://www.cram-system.org/cram, 2019. [Online; accessed 12/11/2019].

[4] ROS- tf. http://wiki.ros.org/tf/Tutorials, 2019-07-18. [Online; accessed 12/11/2019].

[5] M. Beetz, D. Jain, L. Mosenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, Aug 2012.

[6] F. Gravot, A. Haneda, K. Okada, and M. Inaba. Cooking for humanoid robot, a task that needs symbolic and geometric reasonings. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 462–467, May 2006.

[7] D. Leidner, C. Borst, A. Dietrich, M. Beetz, and A. Albu-Schäffer. Classifying compliant manipulation tasks for automated planning in robotics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1769–1776, Sep. 2015.

[8] Lorenz Mösenlechner. *The Cognitive Robot Abstract Machine: A Framework for Cognitive Robotics*. PhD thesis, TECHNISCHE UNIVERSITÄT MÜNCHEN, https://mediatum.ub.tum.de/doc/1239461/1239461.pdf, 19.08.2015.

[9] K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot, and M. Inaba. Humanoid motion generation system on hrp2-jsk for daily life environment. In *IEEE International Conference Mechatronics and Automation, 2005*, volume 4, pages 1772–1777 Vol. 4, July 2005.

[10] Akihiko Yamaguchi and Christopher Atkeson. A representation for general pouring behavior. 09 2015.