
Development and analysis of a library of actions for robot arm-hand systems

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades

“Doctor rerum naturalium”

der Georg-August-Universität Göttingen

im Promotionsprogramm PCS

der Georg-August University School of Science (GAUSS)

vorgelegt von

Mohamad Javad Ain

aus Jiroft, Iran



Göttingen 2016

Referent: Prof. Dr. Florentin Wörgötter

Koreferent: Prof. Dr. Ulrich Parlitz

Weitere Mitglieder der Prüfungskommission:

Prof. Dr. Dieter Hogrefe

Prof. Dr. Xiaoming Fu

Prof. Dr.-Ing. Tamim Asfour

Prof. Dr. Poramate Manoonpong

Tag der mündlichen Prüfung: 16.09.2016

Declaration of Authorship

I, Mohamad Javad AEIN, declare that this thesis titled, 'Development and analysis of a library of actions for robot arm-hand systems' 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:

“Intelligence is what you use when you don’t know what to do.”

Jean Piaget

GEORG-AUGUST UNIVERSITÄT GÖTTINGEN

Abstract

Informatik

3rd Physics Institute

Doctor of Philosophy

Development and analysis of a library of actions for robot arm-hand systems

by Mohamad Javad AEIN

The ability to perform human-like manipulation actions using artificial robots is a major requirement in service robotics. This is a problem related to both high-level symbolic reasoning and low-level control systems. This work proposes a multi-layer framework to fully define and execute a wide range of such actions in a generic and generalizable fashion. We present the details of action definition and execution and collect them into a re-usable software library.

The first contribution of this thesis is definition of high-level and low-level components of actions as well as a clear mechanism to link them in execution. To demonstrate the ability of execution system to generalize on a wide range of actions and objects, a large set of 300 trials is performed. The success rate of each action is calculated and the failure cases are analyzed.

The second contribution is applying the concept of structural bootstrapping to get action parameters from human demonstrations and previous experiences. Here, several human demonstrations obtained by different methods are processed. New instructions are executed based on previous knowledge which enables system to go beyond hard-coded actions.

Last contribution is to integrate the actions with a symbolic decision making framework to benefit from the advantages of the state-of-the-art in planning. Here we deal with grounding symbolic operators of planner to solve complex tasks such as making a simple cucumber salad. We also feedback the faults of execution to the decision-making system which enables learning new operators through a human operator.

Acknowledgements

This project would not have been possible without the help of my family, friends and colleagues. I would like to express my sincere gratitude to my advisor Professor Florentin Wörgötter. He gave me the opportunity to work on an interesting project and supported me along the way. He was always open to discussions and his constructive criticism combined with supportive attitude created a comfortable atmosphere to do research.

I want to thank my colleagues in BCCN Göttingen. Special thanks to Prof. Minija Tamosiunaite, Dr. Eren Erdal Aksoy, Prof. Tomas Kulvicius and Dr. Alejandro Agostini with whom I had the chance to work directly during my project. Together we had many interesting discussions and accomplished several projects. I want to thank Dr. Jan-Matthias Braun not only for his generous technical help and cooperation, but also for his great and positive attitude. I should thank the members of our hard-working computer vision group (Dr. Alexey Abramov, Dr. Jeremie Papon, Dr. Eren Erdal Aksoy, Dr. Markus Schoeler, Simon C. Stein and Simon Reich) for developing our vision system and helping to integrate it with the robotics setup.

I want to also thank all of the former and current members of our group who created a nice and friendly environment and contributed in frequent scientific dialogue especially: Professor Poramate Manoonpong, Dr. Frank Hesse, Dr. Guoliang Liu, Dr. Yinyun Li, Harm-Friedrich Steinmetz, Martin Biehl, Dr. Xiaofeng Xiong, Dr. Christian Tetzlaff, Dr. Michael Fauth, Timo Nachstedt, Fatemeh Ziaetabar, Timo Lüddecke, Juliane Herpich. I want to especially thank Mrs. Ursula Hahn-Wörgötter who was always there in the most difficult times and generously helped me to adapt to life in Germany.

I would like to thank our dear friends Maryam and Morteza who welcomed us in their warm and friendly family and treated me and my wife as their children. I want to thank my parent-in-laws Mitra and Mahmoud and my sister-in-laws Mahsa and Mahya for their constant support, love and motivation. I want to acknowledge the kind spirits of my deceased grandmother "Bibi" Kobra and my father Dr. Mashallah Aein who would feel very proud if they could see me obtaining my PhD.

Last but far from least, I want to give my greatest appreciation to my beloved wife Mahboobeh who always believed in me and stood by me in the most difficult times during this project.

Thank you all!

Sincerely,

Mohamad Javad Aein

Göttingen 2016

Dedicated to my beloved wife Mahboobeh

Contents

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
Contents	vii
List of Figures	x
List of Tables	xvi
Abbreviations	xviii
1 Introduction	1
1.1 Problem Statement	2
1.1.1 Manipulation Action Definition	2
1.1.2 Manipulation Action Execution	3
1.1.3 Learning Manipulation from Demonstration	3
1.1.4 Action Compilation	3
1.1.5 Integrating Planning and Execution	4
1.2 State-of-the-art	5
1.3 Overview and Contributions	9
2 Library of Manipulation Actions	10
2.1 Action definition	10
2.1.1 High-level Action Definition	11
2.1.1.1 Objects Roles	14
2.1.1.2 Semantic Event Chain (SEC)	15
2.1.1.3 Abstract Relations	17
2.1.1.4 Abstract Primitives	19
2.1.2 Low-level Action Definition	20
2.1.2.1 Real objects	20
2.1.2.2 Real Relations	21
2.1.2.3 Real Primitives	24

2.2	Ontology of Actions	31
2.3	Action Execution	35
2.3.1	Mid-level sequencer: Finite State Machine	35
2.3.1.1	States(S)	35
2.3.1.2	Inputs(U)	36
2.3.1.3	Outputs(Y)	36
2.3.1.4	State transition function(f_1)	37
2.3.1.5	Output function (f_2)	37
2.3.1.6	Initial state($S(1)$)	37
2.3.2	Action Execution Engine	39
2.4	Results	42
2.4.1	Hardware	42
2.4.1.1	Robot Arm	42
2.4.1.2	Robot Hand	42
2.4.1.3	Vision System	42
2.4.2	Single Actions	43
2.4.3	Chained Actions	51
2.5	Conclusion	54
3	Bootstrapping Action Execution	55
3.1	Data Acquisition	56
3.1.1	Demonstration by Kinesthetic Guidance	56
3.1.2	Demonstration in Virtual Reality	58
3.2	Bootstrapping from Human Demonstration	60
3.2.1	Calculating the SEC matrix	61
3.2.2	Calculating the Object Roles	62
3.2.3	Calculating the Action Primitives	64
3.2.3.1	Segmentation of Trajectories	65
3.2.3.2	Identify Arm Primitives	72
3.2.3.3	Identify Hand Primitives	74
3.2.4	Combine Multiple Demonstrations	75
3.3	Bootstrapping by Action Compilation	79
3.3.1	Action Data Tables (ADT)	79
3.3.2	Compilation Process	80
3.4	Bootstrapping Results	86
3.4.1	Human Demonstration Results	86
3.4.2	Action Compilation Experiments	88
3.5	Conclusion	89
4	Error handling and Planning	90
4.1	Fault Detection	91
4.2	Planning with Ontology of Actions	95
4.3	Object Replacement	101
4.4	Conclusion	104
5	Short Summary and Final Remarks	105
5.1	Summary	105

5.2 Discussion of Results	106
5.3 Future Work	106
A Ontology of Actions	107
B Execution of Single Actions	119
C Compilation Results	123
Bibliography	125

List of Figures

2.1	Levels of the proposed action definition and execution framework. The high-level components are symbolic and close to human language. The low-level components are in the signal domain and connect to sensors, actuators and control systems. The mid-level fills the gap between them and makes the execution possible.	11
2.2	A sample human demonstration and robot execution of a <i>Put on top</i> action are shown to highlight different action components. At the top, snapshots and segmented images of the human demonstration are shown. Next, a relational graph sequence is computed. Each graph corresponds to one world state (S_1 to S_5). The objects in the scene are recognized and their roles in the action are determined. Abstract spatial relations and their values at each state are shown in the SEC matrix. Here, each row represents a pairwise object relation such as N and T which stand for <i>Not-touching</i> and <i>Touching</i> , respectively. Action primitives at each state are shown at the bottom of the SEC matrix. Finally, some snapshots and segmented images of the robot execution with different objects are shown.	12
2.3	Extracted SEC matrices for 10 single atomic manipulation actions. The abstract spatial relation associated to each SEC row is color coded in which blue and yellow represent <i>Touching</i> (T) and <i>Not touching</i> (N) respectively. The gray color shows either an <i>Absent</i> (A) or a <i>Dont-care</i> relation. Note that the SEC matrix of three actions <i>push by grasp</i> , <i>poke</i> and <i>push by holding</i> are the same, whereas their action primitives and parameters are different. In the action <i>Put on top</i> the <i>primary</i> object is the same as the <i>secondary support</i> , which makes the relations R_5 and R_8 identical. Similarly, relations R_4 and R_7 in the action <i>Take down</i> are identical, since the <i>secondary</i> object is the same as <i>primary support</i>	18
2.4	Calculating the touching relation with our visual perception interface. The red lid is touching the jar and the yellow cup is on top of the bucket (left). The XZY point clouds of the same scene are shown. The red lines indicate a touching relation exists between a pair of objects (right)	22
2.5	Intermediate exteroceptive sensory input in the process of analyzing the spatial relation rules given in Table 2.3. The tactile sensor values are used to detect when an object is grasped by the hand (top). The external force signals are processed to detect the touching event (bottom). Here a contact in Z direction is detected.	23
2.6	Example of trajectories generated by DMPs in 3D space. There are different starting points S_1 to S_8 and all end in the origin. The trajectories S_1 to S_4 are straight lines with zero weights while S_5 to S_8 have half-sine shape.	26

2.7	Examples of periodic trajectories in actions. (a) In the cutting action the following parameters are used to generate a back and forth motion: $a_x = 0. - 008 m$, $a_y = -0.01 m$, $\omega = 1.8 rad/s$. The initial position is $x(0) = -0.6 m$ and $y(0) = 0.7 m$. (b) In the stirring action a circular motion is generated by using the following parameters: $b_x = 0.03 m$, $a_y = -0.05 m$, $\omega = 1 rad/s$. The initial position is $x(0) = 0.187 m$ and $y(0) = 0.55 m$	28
2.8	Example of force control using the <i>arm_exert</i> primitive. The desired force is 1 N in Z direction.	29
2.9	Two pre-shape configurations are used in our system. The power grasp (right) is used for symmetric objects while the precision grasp (left) is for elongated objects.	29
2.10	The action <i>Poke</i> is an example from Category 1	32
2.11	The action <i>Pick and place</i> is an example from Category 2	33
2.12	The action <i>Unload</i> is an example from Category 3	34
2.13	The finite state machine (FSM) for <i>Put on top</i> action is shown. The states of the FSM (S_1 to S_5) are related to the columns of SEC matrix and states of the action. The action starts if initially inputs (real relations) match the first column of the SEC matrix. At each state, the FSM outputs (Y_1 to Y_5) are executed which are the defined primitives of each column. The transition to from state i to $i+1$ occurs when the inputs match the $i+1$ -th column of the SEC matrix. Otherwise, the next state is the same as the current state (loop transitions). This process is continued until reaching the final state which corresponds to the last column of the SEC matrix.	38
2.14	State diagram of the execution engine which controls the execution of actions. This state machine is the main component of the mid-level. The show the diagram more clear, different colors are used for normal execution (blue), error handling (orange), failure (red) and success (green) states and transitions.	39
2.15	The set of objects used in our experiments. There are in total 19 objects in 8 categories: 1-Round fruits 2-Long fruits 3- Cubes 4-Cups 5-Containers 6-Plates 7-Spoons 8-Knives.	43
2.16	Overall success rate of 10 atomic action execution after 30 trials for each.	45
2.17	Success rate of executing actions in each object category. Each action is executed 30 times using different object sets. The ratio of successful trials are shown for each object category (middle columns). For actions involving grasp, the results are separately shown for each grasp type. The overall success rates on each grasp type and average success scores are shown in the last two columns. The values in the last column match the final average accuracy rates shown in Figure 2.16.	45
2.18	Low-level sensory data in a sample <i>put on top</i> action. The position, tactile and force contact signals are shown on the top. All changes in object contact relations are shown in the bottom plot as a color coded SEC matrix. Here, blue and yellow represent <i>Touching (T)</i> and <i>Not touching (N)</i> respectively. The gray color shows either <i>Absence (A)</i> or relations which are not important (dont-care). Some sample snapshots at the bottom show the scene topology at each state of the action.	48

2.19	Low-level sensory data in a sample <i>cutting</i> action. The position, tactile and force contact signals are shown on the top. In the cutting action, a part of the trajectory corresponding to the back and forth motion of knife is zoomed in to show the oscillatory motion pattern. All changes in object contact relations are shown in the bottom plot as a color coded SEC matrix. Here, blue and yellow represent <i>Touching (T)</i> and <i>Not touching (N)</i> respectively. The gray color shows either <i>Absence (A)</i> or relations which are not important (dont-care). Some sample snapshots at the bottom show the scene topology at each state of the action.	49
2.20	Trajectory of robot arm during the <i>Put on top</i> action shown in Figure 2.18.	50
2.21	Trajectory of robot arm during the <i>Cutting</i> action shown in Figure 2.19. .	50
2.22	Robot execution of three chained actions: 1- Taking down the red apple from the box 2- Pushing the box by holding. 3- Putting the green apple on top of the box. From top to bottom are shown the position, tactile, and force sensor data as well as the changes which are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted with numbers showing their order. Black arrows represent temporal interval of each action.	52
2.23	Robot execution of a salad preparation scenario which involves 5 atomic actions: 1- Put on top 2- Cut 3- Unload 4- Pour 5-Stir. From top to bottom are shown the position, tactile, and force sensor data as well as the changes which are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted at the bottom. Black arrows represent temporal intervals for atomic actions.	53
3.1	The operator performs a kinesthetically guided demonstration. The bottle cap is being unscrewed from the bottle.	57
3.2	The data recorded in the demonstrated <i>Unscrew</i> action of Figure 3.1. (a),(b) Cartesian position and orientation of the end effector of the manipulator. (c) The joint angles of the robot hand. (d) The tactile sensor readings during the action.	58
3.3	An example of <i>Insert</i> action demonstrated in the simulated environment.	58
3.4	The data recorded in the <i>Insert</i> example demonstrated in the simulated environment. (a),(b) Cartesian position and orientation of the end effector of the manipulator. (c) The width of the robot hand. (d) The tactile sensor readings during the action.	59
3.5	The process of finding action description from human demonstrations. The demonstrated action is recorded and three main components are calculated: 1- SEC matrix 2-Object roles 3-Action primitives. The trajectories are further analyzed to find the types, arguments and parameters of the primitives.	60
3.6	The relations of objects during the demonstrated <i>Put on top</i> action of Figure 3.8 are shown. The SEC matrix is also derived.	61
3.7	The relations of object during the demonstrated <i>Insert</i> action of Figure 3.3 are shown. The SEC matrix is also derived.	62

3.8	The data recorded from the demonstrated <i>Put on top</i> action are shown. At the top, the snapshots of the demonstration without the operator (second round) is shown. The end effector position and orientation are shown in plots (a) and (b). For the robot hand, the joint angles and tactile sensor readings are shown in plots (c) and (d).	65
3.9	A simple 2D trajectory is segmented by comparing the velocity directions. The velocity at $v(t)$ diverges from the average velocity of segment i , \bar{V}_i , therefore a new segment is created. Green circles show the segmentation points created by using Equation 3.2.	66
3.10	Initial segmentation of arm pose trajectories. The velocities are shown on the top plot. The moving parts of trajectory are shown in the second plot. The resulting segments are shown in the third plot. The last plot shows 3 large segments which are kept after removing small (noisy) motions.	67
3.11	The segmentation of arm pose trajectories using direction of velocities. The 3 large segments (top) are divided into subsegments with different velocity direction. The cosine of angle between velocities are calculated according to Equation 3.1 and a threshold $k = 0.8$ is applied. The subsegments are shown in the third plot. After removing the small subsegments, the final segmentation of pose trajectories are shown in the bottom plot with 10 segments.	69
3.12	The position trajectory is plotted in 3D space and the start and end points are marked. The average velocity of each segment is superimposed at its middle point. The trajectory for this action is divided to 10 segments.	70
3.13	Segmentation of the trajectories of robot hand. The velocities are calculated (top plot) and the <i>hand-is-moving</i> flag is shown (second plot). Based on this flag, the trajectory is segmented (third plot) and after removing the small segments, 3 segments are detected.	70
3.14	The 3 segments of Figure 3.13 can be further divided into sub-segments. The cosine of velocities is calculated and the segmentation points are calculated based on Equation 3.2 with $k = 0.3$. (second plot) The resulting sub-segments are shown in the third plot. After removing the small sub-segments, the final segmentation of hand trajectory is shown in the last plot.	71
3.15	The direction, distance and combined measures for the <i>Put on top</i> example. The object with the lowest combined measure is selected as the argument of each primitive.	74
3.16	The primitives of hand are extracted by combining data from joint angles and tactile sensors. Segmentation of joint trajectories reveals the number and time of primitives (top). The tactile events (bottom) help us to distinguish between <i>hand_preshape()</i> , <i>hand_grasp()</i> and <i>hand_release()</i> primitives.	75
3.17	The obtained arm and hand primitives of the <i>Put on top</i> action demonstrated by human operator. The top and middle plots shows the hand and arm primitives. Start and end times are shown by green and red circles. The time of each primitive is shown by a blue line. The type of primitives are shown in the Y-axis. For the <i>arm_move</i> primitives, the argument object is shown in the Y-axis. (free, main, primary and secondary). In the last three plots, the observed relations of objects are shown to give a better description of the action.	76

3.18	Human Demonstrated <i>Put on top</i> actions with different objects.	78
3.19	Primitives of <i>Put on top</i> action by combining the results of all demonstrations.	78
3.20	An example of a part of an ADT file for an <i>Unscrew</i> action. Actual ADTs have more entries which are omitted here to save space.	80
3.21	Block diagram of compilation process. We use two databases in this process. First is the Action ontology introduced in Section 2.2. The second is the ADT database which is the collection of our executed actions in the ADT format.	81
3.22	A scene with some objects to test the compilation algorithm. We want to execute the following instruction: Shake the plastic bottle and put it on the tray.	81
3.23	The definition of action <i>Shake</i> which is associated to the output of parser, is fetched from the ontology. We use its SEC, abstract objects and primitives in the compilation process	83
3.24	The results of primitive detection is evaluated by comparing the results with the human-generated ground truth. The accuracy and precision of both kinesthetically guided experiments (top) and virtual reality experiments (bottom) are shown.	87
4.1	Error handling after failure in grasping an object. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4,5- When the manipulator approaches to grasp the apple, we move it to cause the grasp to fail. 6- The robot hand opens and the robot arm moves up waiting for a new perception. 7- New perception of the objects by vision system. 8,9- Approaching the apple in its new position and performing a grasp.	92
4.2	Error handling after failure in <i>Put on top</i> action. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4- When the manipulator tries to put the apple on top on the board, we move the board to cause the failure. 5- The manipulator moves up and waits for a new perception. 6- New perception of the objects by vision system. 7,8,9- The manipulator puts the apple on the board in the new position.	93
4.3	Error handling after failure in the push by holding action. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4,5- When the manipulator tries to hold the apple from top and push it. However, this fails due to inaccuracy in perception and the geometry of the apple. 6- The manipulator moves up and receives a new perception of the objects from the vision system. 7,8,9- The manipulator tries the pushing again and this time it succeeds.	94
4.4	Error handling after the grasped objects slips through the robot hand. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4- The manipulator approaches the apple and grasps it successfully. 5- The grasped object is taken out of the robot hand to cause the error. 6,7- The robot hand opens and the manipulator retracts. 8- New perception of the objects is received from the vision system. 9- The grasp is repeated successfully.	94
4.5	The initial and goal states for a simple planning problem.	99
4.6	General diagram for plan generation and object replacement. (Reference: [1])	100

4.7	ROAR is an intelligent database which updates its content. (Reference: [1])	101
4.8	The modified scene for salad making in which the cucumber is missing. There are two additional objects, the jar and the banana.	102
4.9	The modified salad making which uses the banana instead of cucumber.	102
4.10	The execution of chained actions: Unload and Stir by using the spoon.	103
4.11	The replacement of the spoon by the knife. The plan is updated to use knife instead of spoon.	103

List of Tables

2.1	Object roles defined based on spatial relations. Each role is defined and the constraints on the relations are presented. Note that main object is defined with regard to the manipulator, unless the action is performed using a tool. In this case, the main object is defined with regard to the tool.	16
2.2	Abstract relations and their attributes for the action “ <i>putting a bucket on a box</i> ” shown in Figure 2.2.	19
2.3	Rules for detecting the spatial relational changes during action execution. Note that $\ manip, O_i\ $ represents the Euclidean distance between manipulator end-effector and position of object O_i . Similarly, $\ O_i, O_j\ $ is the distance between objects O_i and O_j . O_i and O_j are any pair of objects in the scene. $contact_z$ means that the sensor sensed a contact in Z axis, while $!contact_z$ means there is no such contact. $grasped$ means the hand has grasped some object, while $!grasped$ means the hand is empty. The parameters \mathcal{D}_{ij} are the distance thresholds to decide whether two objects are touching or not.	22
2.4	Rule consideration for detecting spatial object relations (See Table 2.3) and Figure 2.3.	23
2.5	Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category.	31
2.6	List of 10 atomic actions stored in the library and also used in experiments introduced in Section.2.4.2. The last two columns show sample objects used in each action.	44
3.1	The object roles of two examples are obtained by analyzing the object relations. The <i>Put on top</i> example is shown in Figure 3.8 and the <i>Insert</i> action is performed in the simulation environment of Figure 3.3.	64
3.2	Translational and angular distances of each segment of arm pose trajectory.	72
3.3	The primitives are 6 demonstrations of <i>Put on top</i> action are compared at each state. The entries of table are the number that each primitive type and argument appeared in each demonstration. The primitives which are more frequently repeated across different demonstrations, are kept in the general description of the <i>Put on top</i> action.	77
3.4	Output of parsing of the example instruction	82
3.5	List of candidate primitives from all actions in the ontology which are similar to primitives of <i>Shake</i> action. The actions for which we have at least one example in ADT database are marked with an asterisk (*)	84

3.6	For the first primitive of the desired <i>Shake</i> action, the following candidates exist in the ADT database. The first three options all have the same <i>main</i> and <i>primary</i> objects.	85
3.7	The list of actions demonstrated by human operator. The method of demonstration, the action name and the roles of objects in the action are specified.	86
3.8	Compilation results for the instruction <i>Shake the plastic bottle and put it on the tray</i> . For each primitive we indicate the action in the ADT database and the primitive from which the parameters are replaced. . . .	88
A.1	Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category.	108
B.1	Pick and place	119
B.2	Push with Grasp	119
B.3	Push by Holding	120
B.4	Poke	120
B.5	Put on Top	120
B.6	Take Down	121
B.7	Push apart by Holding	121
B.8	Push together by Holding	121
B.9	Cutting	122
B.10	Stirring	122
C.1	Instruction: <i>Push the bottle away from the jar</i>	123
C.2	Instruction: <i>Take the jar and place it into the box</i>	123
C.3	Instruction: <i>Drop the bottle into the wastebasket</i>	123
C.4	Instruction: <i>Insert the spoon into the jar</i>	124
C.5	Instruction: <i>Unscrew the lid from the mug</i>	124
C.6	Instruction: <i>Invert a jar</i>	124

Abbreviations

AI	Artificial Intelligence
SEC	Semantic Event Chain
FSM	Finite State Machine
DMP	Dynamic Movement Primitive
ROS	Robot Operating System
OROCOS	Open Robot Control Software
ADT	Action Data Table
PKS	Planning with Knowledge and Sensing

Chapter 1

Introduction

There is a growing need for intelligent service robots in different non-industrial sectors. Examples are household robots able to help in cleaning and cooking, robots which are able to interact with children and service robots able to help the seniors with simple everyday-life tasks. This need is one factor contributing to the growth of the field *service robotics*. One major requirement for a service robot is the ability to manipulate objects found in human environments.

It is interesting that almost all the robots developed by experts in AI and robotics, perform poorly in manipulating objects and performing tasks compared to a 5-year-old child, let alone an adult who is professional in his/her field. The manipulation ability of humans come from their high-performance and flexible sensors and actuators combined with the unmatched processing capabilities of their brain.

In this thesis, we study the problem of replicating human-like manipulation actions with artificial agents. Such agents usually take the form of human-like (anthropomorphic) robotic manipulators which are able to move, push, grasp and carry around objects. Recent advances in building redundant, light-weight and compliant robot arms (such as KUKA LWR IV, Universal UR5, etc.) and dexterous robot hands (Schunk SDH2, Robotiq hand, etc.) provide the necessary hardware to achieve this goal. Although these manipulators are still far away from dexterity of the human arm and hand, they are sufficient for building working systems.

Achieving human-like manipulation skills requires contributions from several computer science and AI fields (e.g. planning and machine vision) as well as other fields like robotics

(e.g. path planning and grasping) and control engineering (e.g. position and force control). In this thesis we try to contribute to the sub-field of manipulation planning, learning and execution. Our specific goals are summarized as follows:

- To employ AI and robotics techniques to establish a framework for definition and execution of manipulation actions.
- To develop mechanisms to bootstrap existing knowledge by using human demonstrations and previous knowledge.
- To integrate the designed execution framework with other existing modules i.e. a symbolic planner and computer vision.

The rest of this chapter is organized as follows: The problems which we want to deal with are stated in 1.1. A review of the state-of-the-art in the literature on these topics is provided in 1.2. The contributions of this thesis are summarized in 1.3 to conclude this chapter.

1.1 Problem Statement

Previously we stated the goals of this thesis. Here we will expand these goals to several sub-problems and discuss our proposed solutions in the remaining chapters.

1.1.1 Manipulation Action Definition

How to define a manipulation actions?

There have been two main approaches to this problem based on symbolic and geometric (sub-symbolic) representations. The symbolic approach is most common within classic AI and natural language communities. Engineers and roboticists usually prefer more geometric approaches dealing with low-level signals.

Both approaches have advantages and drawbacks. The symbolic approach it is more intuitive in defining tasks and relating them to humans. It also generates a discrete state space which makes planning tasks more tractable compared with the signal space which is of continuous nature. However, the major problem is the *grounding* of these

symbols in the environment. In signal space the main problem is to find a small subset of features for manipulation actions. Two demonstrations of the same pick and place action could look totally different in signals space, which makes it difficult to find a conjoined symbolic representation for this.

More recent approaches, including our approach, try to combine both approaches to have the benefits of the both. However this is far from a trivial problem. It is not clearly known what constitutes the essence of an action and what parameters should be adapted to deal with different situations. The question remains to separate “core” action parameters from “situation-dependent” ones.

1.1.2 Manipulation Action Execution

How to execute manipulation actions?

This problem is much related to the previous question, since the representation of actions affects their execution. The key question in execution is the ability to generalize over different robot embodiments, actions, objects and poses. The execution should keep the essence of action (e.g. pick and place) while being able to adapt the low-level signals to new situations.

1.1.3 Learning Manipulation from Demonstration

How to learn actions and action parameters from demonstrations?

This question has both theoretical and empirical importance. There are many learning methods for this but the most common is ... Theoretically it is challenging since it is in general not known how the symbolic representation of an action is related to the low-level trajectories. It is specially difficult when the demonstration is done in a cluttered scene in which the effect of surrounding objects complicates the interpretation of signals. In practice, it is important to develop algorithms applicable to a general robot platform and deal with sensor noise and uncertainty.

1.1.4 Action Compilation

How to execute actions in new situations by using existing experience?

It is desirable to use previous experience and apply it in novel situations, for which there is no pre-programmed plan. This requires a data structure to store and retrieve the experience of the robot. In addition we need an algorithm to systematically combine the proper parameters from old experiences and to *compile* new actions.

1.1.5 Integrating Planning and Execution

How to use higher level knowledge (Planners) to solve more complicated tasks?

We want to use the existing symbolic decision making (planning) systems to enhance the performance of action execution. This enhancement is achieved in two ways. First by chaining the single actions and creating a longer sequence to perform more complex tasks like making a salad. Second, the planner is used to deal with the faults and failures of the execution. This requires detecting the failures at runtime and reporting them to the higher level planner. The planner then analyzes the fault and makes the best decision.

1.2 State-of-the-art

For each of the problems mentioned in 1.1 a review of the existing literature will be presented.

- Manipulation Action Definition

There exists a large corpus of work in action representation and execution [2–6]. Two distinct approaches are commonly preferred in order to represent and execute actions, one at the trajectory level [2] or the other at the symbolic level [7]. The former gives more flexibility for the definition of actions, while the latter defines actions at a higher level and allows for generalization and planning.

For trajectory level representation there are several well established techniques: Splines [3], Hidden Markov Models (HMMs) [4, 8], Gaussian Mixture Models (GMMs) [5], Dynamic Movement Primitives (DMPs) [2, 9]. With trajectory level encoding, one investigates or learns different complicated trajectories, but it is difficult to use them in a “more cognitive sense”. Generalization of the observed trajectories is the main *breaking point* here, since even the same action can be demonstrated by following various trajectories.

High-level symbolic representations many times use graph structures and relational representations, e.g. [10–12]. Alternative methods, such as in [13]), describe a syntactic approach for learning robot imitation by capturing underlying task structures in the form of probabilistic activity grammars. These approaches give compact descriptions of complex tasks, but they do not consider execution-relevant motion parameters (trajectories, poses, forces) in great detail. In this work, our high-level action descriptor is based on the concept of Semantic Event Chains (SECs) introduced in [12]. SECs are generic action descriptors that capture the underlying spatio-temporal structure of continuous actions by sampling only decisive key temporal points derived from the spatial interactions between hands and objects in the scene. The SEC representation is invariant to large variations in trajectory, velocity, object type, and pose used in the action. Therefore, SECs can be employed for the classification of actions as demonstrated in various experiments in [12, 14–17]. In contrast to works in [10, 11], we have already shown in [15] and

[18] that actions encoded by SECs can be executed once the low-level data (object positions, trajectories, etc.) are provided.

The concept of semantic event chains has also been successfully utilized and extended by others [19–23] for monitoring or execution purposes. [19] addressed the reproduction of the obtained human action sequence by parameterizing semantic conclusive sub-actions with the SEC framework. Active learning of goal directed manipulation sequences, each was recognized using semantic similarities between event chains, was presented in [22]. Scene graphs used in SECs were also represented with kernels in [20] to further apply different machine learning approaches. Additional trajectory information was used in [21] to reduce noisy events in SECs. In [23], SEC-like graphs are used to recognize categories of actions based on the consequences. All these studies confirm the scalability of the event chains to various monitoring tasks.

Many times trajectory-level descriptions of actions, object properties, and high-level goals of the manipulation are brought together through STRIPS-like planning [7], resulting in operational although not very transparent systems. The approach in [24] attempts to integrate a symbolic action representation and planner with a motor skill learner. The robot learns the goal of the human demonstrated actions by using the Visuospatial Skill Learning (VSL) method which produces symbolic predicates. Such predicates are directly fed to a standard planner to encode skills in a discrete symbolic form. The proposed framework also considers sensorimotor skills, such as the followed trajectory information from the observed action. In [25], a symbolic description of robot tasks is constructed from continuous signals. The work on [26] presents a probabilistic-flow-tubes representation of actions and a method to recognize them in human demonstrations.

In contrast to these works, we do not require an additional symbolic planner since SECs provide a fully observable state sequence which substitutes the symbolic planner. Such approaches are also not evaluated on long and complex human manipulation actions. Thus, how to bring these trajectory and symbolic levels together remains a big challenge in robotics.

In [27] an overview of manipulation action literature and definitions are provided. Our hierarchical definition of actions based on action primitives belongs to the same line of research.

- Manipulation Action Execution

There are also many works concentrating on executing manipulation actions using robots. In [28] a finite state machine is designed to execute a pouring action. [29] used guided motor primitives (GMP) to perform painting and cleaning tasks with a simulated robot. Another approach to deal with the signal-symbol gap is to combine motion and task planning such as the work of [30]. They generate trajectories for tasks, like pick-up and put-down, to solve problems in different domains. While the existing works show good results, they are usually limited in the number of actions and objects used. One of the goals of our study is to develop many actions and test them on many objects.

In Beetz et al. [31] a software toolbox called Cognitive Robot Abstract Machine (CRAM) is proposed. This toolbox provides tools for develop programs for cognitive robots specially mobile manipulation. In [32] actions are executed by sequencing the action primitives defined by DMPs.

- Learning from Demonstration

In learning from demonstration (LfD) the robot tries to learn how to perform a task from examples provided by a human operator. The operator may interact with the robot through different mechanisms: tele-operation, motion tracking, virtual reality or kinesthetic guidance. However, the goal is to autonomously learn and generalize through the provided examples and apply the skill in new situations.

There is a large body of research about this paradigm, also by the names programming by demonstration (PbD) [7] and imitation learning. Here we summarize some of the highlights related to the current thesis. In [33] an architecture is proposed to integrate machine learning and LfD. They studied some problems of LfD such as generalization of a task, given a set of examples. The operator has a key role in their architecture. The work of [34] provides a framework for learning tasks from multiple demonstrations. They use additional cues in the demonstrations to have the attention of learner in the decisive moments. In [35], symbolic descriptions are extracted from human demonstrations by segmenting the hand and arm trajectories. The actions are described by macro operators (MO) and elementary operators (EO) in a hierarchy. In [36] a learning method is proposed to enable a human teacher to shape robot behavior. Another related work is [37] in which

task precedence graph (TPG) is introduced to model the human demonstrated actions and reproduce them. In [38] a bottle opening task is segmented into different control strategies and the contribution of each part is weighed.

Many of these approaches need to segment the trajectories from a demonstrated action. For example in [39] an autonomous framework to segment robot trajectories are proposed based on GMMs. In [40] uses HMMs and DMPs to segment a demonstrated task and generalization over repeated trials.

- Integrating Planning and Execution

Classic AI methods such as famous SHRDLU [41] tried to represent manipulation actions by symbolic descriptions. However, these methods can not deal with the complexity of real-world situations. To benefit from their reasoning capabilities, the operators of a symbolic planner should be grounded in the real world. There have been many attempts to designs an integrated system. In [42], a framework is proposed to deal with the mobile manipulation problem. They decompose the problem into symbolic and geometric parts. For an intuitive goal description, they use a classical symbolic planner. On the other hand, to achieve collision-free trajectories, a probabilistic road-map planner is used. However, the manipulations contain only simple pick-and-place and grasps.

In [43] a mapping between geometric states and logical predicates is learned by using labeled examples. Their mapping can be used in both directions, from symbol to signal and vice versa. They could solve simple tasks in a limited domain. However, the planner can not solve more complex tasks and the system fails.

1.3 Overview and Contributions

The contributions of this thesis are summarized as follows:

1. A novel definition of manipulations actions using semantic event chains, object roles and primitives. Similar actions are divided into three main categories.
2. A novel method for execution of actions generalizable to different robot embodiments, actions, objects and poses.
3. Incorporating a structural bootstrapping concept to obtain the parameters of actions from human demonstrations.
4. Applying structural bootstrapping to re-using action parameters from previous experience (action compilation).
5. Integrating a symbolic decision-making framework with the manipulation actions.

Chapter 2

Library of Manipulation Actions

In this chapter we want to propose a library of manipulation actions based on the idea of semantic event chains (SEC). The SEC framework has first been proposed in [14] and applied successfully in action recognition and classification [12, 16]. It is also used to execute simple actions on simulated robots in [15]. However a concrete framework to execute actions based on the SEC concept was missing.

The library of actions provides both a full definition of actions and a mechanism to execute them on robot manipulators. The actions in the library are also collected into an *ontology* of actions with categories and sub-categories.

The first step to this end is to give a complete and detailed definition that covers both symbolic and sub-symbolic aspects of actions. This is done in Section 2.1. In Section 2.2 the actions are enumerated and categorized. Section 2.3 is dedicated to the execution mechanism. The performance of this framework is tested through a large set of experiments presented in Section 2.4. Finally, a discussion of the results and conclusions are given in Section 2.5.

2.1 Action definition

As illustrated in Fig.2.1, our proposed perception-action framework involves three main levels: *high-*, *mid-*, and *low-level* action units. In this section we will provide the detailed description of high and low levels together with their components.

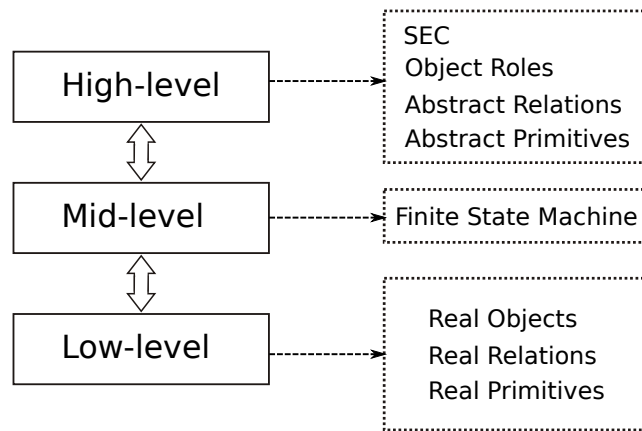


FIGURE 2.1: Levels of the proposed action definition and execution framework. The high-level components are symbolic and close to human language. The low-level components are in the signal domain and connect to sensors, actuators and control systems. The mid-level fills the gap between them and makes the execution possible.

2.1.1 High-level Action Definition

Here we will give a high-level action definition to encode the semantics of manipulations. Note that at this level, definitions are mainly symbolic (abstract) and close to human descriptions.

Take the example of a manipulation action “*put a bucket on a box*”. Figure 2.2 shows some sample frames from human demonstration. This simple action may be described by a human as follows:

1. *Approach* the bucket
2. *Grasp* the bucket
3. *Lift* the bucket from table
4. *Place* the bucket on the box
5. *Release* the bucket

This description is by no means unique. One could easily describe the same action in different words, with different number of steps and details. However, one could still extract some common and descriptive properties from such a naive description:

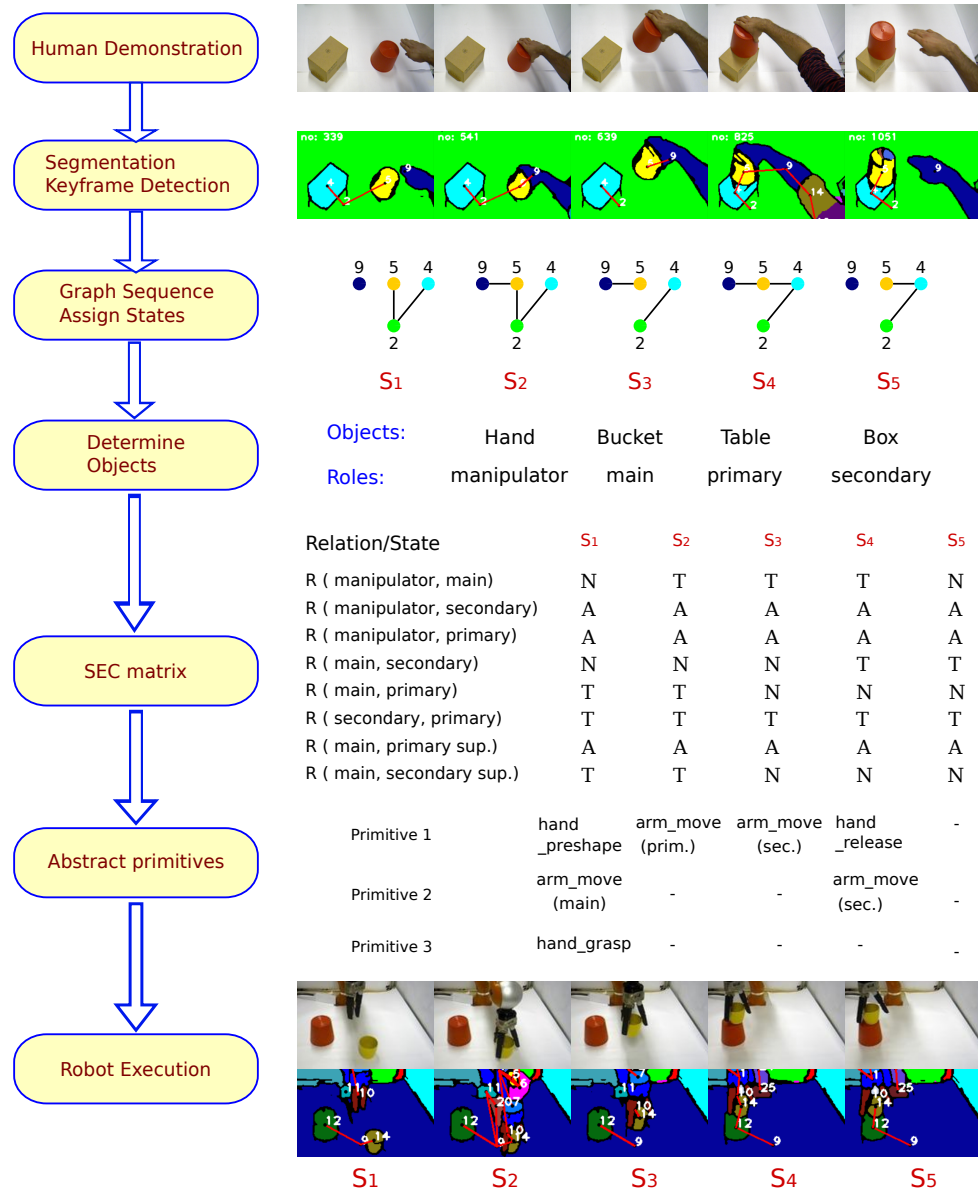


FIGURE 2.2: A sample human demonstration and robot execution of a *Put on top* action are shown to highlight different action components. At the top, snapshots and segmented images of the human demonstration are shown. Next, a relational graph sequence is computed. Each graph corresponds to one world state (S_1 to S_5). The objects in the scene are recognized and their roles in the action are determined. Abstract spatial relations and their values at each state are shown in the SEC matrix. Here, each row represents a pairwise object relation such as N and T which stand for *Not-touching* and *Touching*, respectively. Action primitives at each state are shown at the bottom of the SEC matrix. Finally, some snapshots and segmented images of the robot execution with different objects are shown.

- **Property 1:** The definition is still valid even if the manipulated objects are (within reason) altered.
- **Property 2:** The action (here *Put on top*) can be broken into a sequence of smaller sub-actions (primitives) such as *Approach* and *Grasp*.

- **Property 3:** There are conditions to end one primitive and start with the next. In the above example these conditions are not spelled out explicitly.
- **Property 4:** As humans, we intuitively know how to perform these primitives, although our exact movements are only then produced when we see the objects and are adapted to the scene context while we perform the action.

The main features that we use here to describe a scene are the touching relations between its objects. During a manipulation action, these touching relations change from some initial state to a final state. A manipulation action is, therefore, represented by a sequence of changes in touching relations of the objects.

Our approach to represent and execute manipulation actions with robots has the following fundamental properties: We introduce a generic high-level definition of actions which is independent of the manipulated objects in the action (*Property 1*), and consists of a sequence of symbolic primitives (*Property 2*). The conditions to start and end each primitive are defined by considering the touching relation between objects in the action (*Property 3*). We also store the default action descriptive parameters (e.g. trajectory) to execute actions at the high-level with symbolic definitions. When novel physical objects are observed at each specific instance of an action, these parameters are adapted according to the situation to generate the required movements (*Property 4*).

To fully satisfy these four properties in our high-level action definition, we benefit from the *ontology of manipulation actions* introduced in [44]. This ontology structures human demonstrated manipulation actions, e.g. *putting a bucket on a box*, as sequences of spatio-temporal interactions between objects (including the manipulator) in the scene by using the concept of Semantic Event Chains (SECs) presented in [12].

Our hypothesis claims that the most important action-related information are encoded at the decisive temporal anchor points, i.e. when touching relations between objects and hands change. This ontology suggests about 30 fundamental and unique manipulations that allow complex and chained activities, e.g. *“making a salad”* or *“preparing breakfast”*.

The ontology also introduces four constraints on the definition of manipulation actions, which are stated as follows:

- **Constraint 1:** The action is performed by one hand. This is true for most human actions, since the second hand is usually used only as a support.
- **Constraint 2:** The hand touches exactly one object in the course of the action and does not purposefully touch other objects in the scene unless the current action ends.
- **Constraint 3:** The hand is free at the beginning and at the end of the action.
- **Constraint 4:** The action must lead to some changes in the touching relations between objects and hands (e.g. human or robot hand). In other words, the hand must interact with at least one object.

From the first two constraints one concludes that in each action there are at least two entities: one hand and one object which is directly touched by the hand. This fact will be used in section 2.1.1.1 to define object roles. The second and third constraints together define actions in a way that they can not be further split into shorter actions. The last constraint assures that there is at least one change in the touching relations. This is essential since the whole framework relies on the touching relations between objects.

We extend this ontology by incorporating explicit object roles and adding action primitives. This extended ontology is presented in more detail in Section.2.2. In the rest of this section, we will describe several components of the high-level action definition which are required to reach these descriptive properties.

For more details on the action ontology refer to [45] and Section.2.2 of this thesis.

2.1.1.1 Objects Roles

There exist many objects in the real world and actions can be performed with different sets of object combinations. It is, however, not practical to define a separate action for each possible object set. Instead, as stated in *Property 1*, we represent manipulation actions in a generic way to make them applicable to any novel object. For this, we label objects by their roles exhibited in the action. First of all, recalling *Constraint 1*, we need an actor to perform the action, which is here called *manipulator*. As stated in *Constraint 2*, there exists exactly one object that is directly manipulated by the manipulator. This

object is called *main*. Optionally, there are other objects in the action, which interact with the *main* object in different ways.

The object roles can be better explained in an example. In the action “*putting a bucket on a box*” depicted in Figure 2.2, the human hand is the *manipulator*, the bucket which is directly touched by the hand is the *main* object. There are two more objects whose relations with *main* change in the action: table and box. The relation of *main* and the table changes from touching T to not-touching N . We call such objects *primary* or *source* object. Conversely, the relation of *main* and the box changes from not-touching N to touching T . These objects are called *secondary* or *destination* object.

The complete list of object roles with their definitions are shown in Table 2.1. Some roles are defined by the changes in relations, like *primary* and *secondary*, while others (like support objects) are defined based on constant touching relations. For instance *secondary support* is the object on which the *secondary* object is located. In the above example, the table plays also the role of *secondary support*. Note that not always all objects are needed to define an action.

The role of objects are automatically detected with the method described in [46], which explores the temporal evolution of spatial object relations embedded in SECs.

2.1.1.2 Semantic Event Chain (SEC)

At the highest symbolic-level, actions are represented by the concept of Semantic Event Chains (SEC) which captures the essence of an action by focusing on the touching relation of object. These relations are calculated by employing computer vision techniques described in [12, 47]. A summary of this process is shown in Figure 2.2 along with the *Put on top* example. To calculate the SEC representation, an image sequence of an observed action is first represented by 3D image segments, each of which corresponds to one object in the scene and is consistently tracked during the action. Each frame in the sequence is then converted into a graph: nodes represent tracked segments, i.e. objects, and edges indicate the contact relation between a pair of objects. By employing an exact graph matching method, the continuous graph sequence is discretized into decisive main graphs, i.e. “states”, each of which represents a topological change in the scene. The extracted main graphs form the core skeleton of the SEC, which is a matrix where rows

TABLE 2.1: Object roles defined based on spatial relations. Each role is defined and the constraints on the relations are presented. Note that main object is defined with regard to the manipulator, unless the action is performed using a tool. In this case, the main object is defined with regard to the tool.

Object Role	Description	Relation Constraints
manipulator	The object that performs the action	Not touching anything at the beginning and the end of action. During the action, it touches at least one object.
main	The object which is directly in contact with the manipulator (tool)	Not touching the manipulator (tool) at the beginning and the end of action. It touches the manipulator (tool) at least once.
primary	The object from which the main object separates	Initially touches the main and makes a T to N transition
secondary	The object to which the main object joins	Initially does not touch the main and makes a N to T transition
load	The object which is indirectly manipulated	Does not touch the manipulator. During the action leaves the main and touches the container or vice versa.
container	The object whose relation with load changes and it is not the main object	Touches or untouches the load object
main support	The object on which the main object is located	Touching the main object all the time
primary support	The object on which the primary object is located	Touching the primary object all the time
secondary support	The object on which the secondary object is located	Touching the secondary object all the time
container support	The object on which the container is located	Touching the container all the time
tool	The object which is used by the manipulator to enhance the quality of some actions	Grasped by the manipulator at the beginning of action and released at the end

are the spatial relations between object pairs in the scene. Each column of the SEC matrix is interpreted as a state of the scene, which is the combination of object relations when a new main graph occurs.

Possible spatial relations in the SEC matrix are *Not touching* (N), *Touching* (T), and *Absence* (A), where N corresponds to two spatially separated objects, T represents objects that touch each other. The value A occurs when there exists no information about the relation, e.g. one object is not visible in the scene.

In a SEC, the progress of the action from the beginning to the end is stored in a compact way. In addition, the SEC matrix is invariant to large variations in trajectory, velocity, object type, and pose used in the action and, therefore, remains the same for different instances of the same action.

Figure 2.2 shows a *Put on top* action from human demonstration to robot execution. The snapshots of the demonstration are shown together with the tracked segments (colored regions) and main graphs. The objects in the scene and the extracted SEC matrix are shown with the corresponding states and primitives. At the bottom, the snapshots and tracked segments of the robot execution are depicted.

Figure 2.3 depicts the event chain patterns of different actions in the library as color coded images. These SEC patterns are stored as high-level action descriptors in the action library. Although SEC patterns are very distinctive, some are semantically identical as in *Push by grasp*, *Poke*, and *Push by holding* actions. This semantic similarity is natural since those actions have the same changes in the touching relation of objects. However, they have different primitives with different object poses, trajectories, and force parameters which are not captured by SECs.

This action descriptive object-, trajectory- and force-information is separately stored as primitives (see Sections 2.1.1.4 and 2.1.2.3).

2.1.1.3 Abstract Relations

We continue with analyzing the spatial relations between each abstract object pair, e.g. between the *manipulator* and the *main* object. Table 2.2 lists the abstract relations for the action “*putting a bucket on a box*”, previously shown in Figure 2.2.

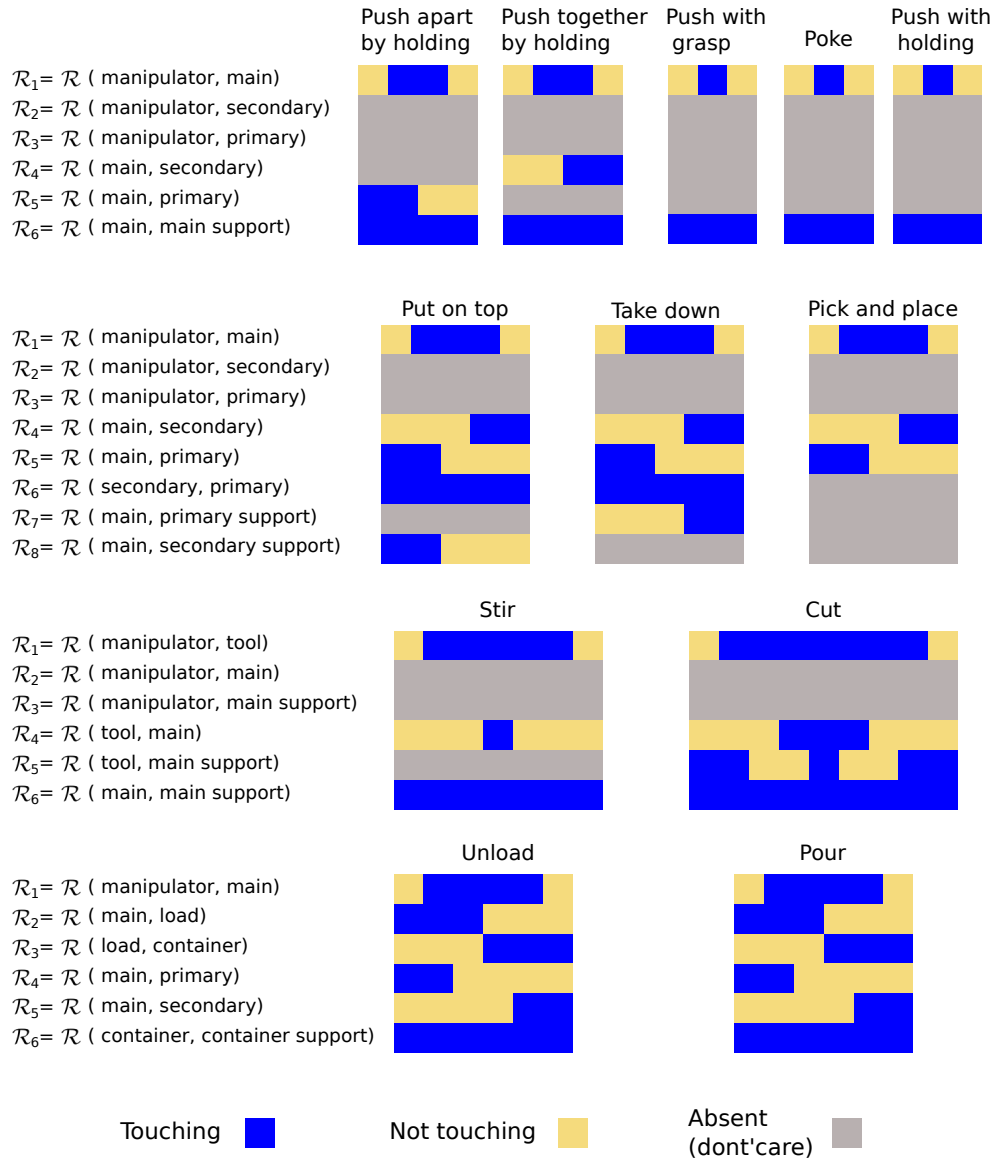


FIGURE 2.3: Extracted SEC matrices for 10 single atomic manipulation actions. The abstract spatial relation associated to each SEC row is color coded in which blue and yellow represent *Touching* (T) and *Not touching* (N) respectively. The gray color shows either an *Absent* (A) or a *Dont-care* relation. Note that the SEC matrix of three actions *push by grasp*, *poke* and *push by holding* are the same, whereas their action primitives and parameters are different. In the action *Put on top* the *primary* object is the same as the *secondary support*, which makes the relations R_5 and R_8 identical. Similarly, relations R_4 and R_7 in the action *Take down* are identical, since the *secondary* object is the same as *primary support*.

Each relation has a *type* and a *value*. The *type* of a relation is determined by the importance and variation of that relation throughout the action. For example, for the action in Figure 2.2, the relation between the *manipulator* and the *primary* is always not-touching and does not affect the outcome of the action. The type of such relations is *don't-care*.

TABLE 2.2: Abstract relations and their attributes for the action “*putting a bucket on a box*” shown in Figure 2.2.

Relation Name	Abstract Relation	Real Relation	Type
\mathcal{R}_1	$\mathcal{R}(\text{manipulator,main})$	$\mathcal{R}(\text{hand,bucket})$	Variable
\mathcal{R}_2	$\mathcal{R}(\text{manipulator,secondary})$	$\mathcal{R}(\text{hand,box})$	Don't-care
\mathcal{R}_3	$\mathcal{R}(\text{manipulator,primary})$	$\mathcal{R}(\text{hand,table})$	Don't-care
\mathcal{R}_4	$\mathcal{R}(\text{main,secondary})$	$\mathcal{R}(\text{bucket,box})$	Variable
\mathcal{R}_5	$\mathcal{R}(\text{main,primary})$	$\mathcal{R}(\text{bucket,table})$	Variable
\mathcal{R}_6	$\mathcal{R}(\text{secondary,primary})$	$\mathcal{R}(\text{box,table})$	Constant
\mathcal{R}_7	$\mathcal{R}(\text{main,primary support})$	-	Absent
\mathcal{R}_8	$\mathcal{R}(\text{main,secondary support})$	$\mathcal{R}(\text{bucket,table})$	Variable

Other relations, which are important for an action, have *variable* or *constant* types. For example, the relation between the *manipulator* (i.e. hand) and the *main* object (i.e. bucket) in Figure 2.2 is *variable* since it naturally alters during the action. The variable relations encode the dynamics of the action. On the other hand, the relation between the *secondary* object (i.e. box) and the *primary* (i.e. table) remains constantly touching, and hence is *constant*. We note that such constant relations highlight the necessary pre-conditions to perform an action and any unexpected change in these constant relations implies an error in the execution.

2.1.1.4 Abstract Primitives

As stated in *Property 2* in Section 2.1.1, an action can be divided into several sub-actions or primitives. In our approach we define the following abstract primitives:

- *arm_move(object)*: The robot arm moves to a pose relative to *object*.
- *arm_move_periodic()*: The robot arm moves periodically.
- *arm_rotate()*: The robot arm rotates around a given axis.
- *arm_exert()*: The robot arm exerts a force.
- *hand_preshape()*: The robot hand moves to a certain pre-shape.
- *hand_grasp()*: The robot hand performs a grasp.
- *hand_release()*: The robot hand releases the already grasped object.

These abstract primitives correspond to the basic functions of the robot manipulator, which can be implemented in many different ways. Our implementations will be presented in Section 2.1.2.3. The focus of our work is, however, not a specific implementation, but rather we would like to propose a way to combine them to seamlessly perform actions. In our approach a state transition in the SEC, i.e. a change from one column to the next, needs at least one of these unique primitives. Thus an action is performed when all of its primitives are sequentially executed while the relations change according to the SEC matrix.

In Figure 2.2, the necessary primitives associated with each column of the SEC matrix are shown. The reason of having multiple primitives is that sometimes more than one primitive is required to induce the desired change in the spatial relation. For example, the combination of *arm_move(main)* and *hand_grasp()* primitives is necessary to change the relation of *manipulator* and *main* from *N* to *T*.

2.1.2 Low-level Action Definition

In this section, the abstract components of the high-level definition are related to their real-world counterparts at the signal-level. This includes defining objects in the real world, calculating their spatial relations from the sensor data, and implementing low-level primitives such that proper commands are sent to the robot arm and hand control systems. In the rest of this section, these elements are described in more detail.

2.1.2.1 Real objects

In real world experiments, abstract objects (i.e. *manipulator*, *main*, *primary*, etc.) are instantiated by real objects in the scene. For the “*putting a bucket on a box*” example depicted in Figure 2.2, these objects are hand (*manipulator*), bucket (*main*), table (*primary*), and box (*secondary*). We need to identify the real-world objects in the signal space in order to perform the low-level primitives.

For this task, we use our modular computer vision architecture described in [47], which segments each object in the scene by employing the color and depth cues fed from the RGB-D sensor. We further apply the instance based object recognition method from [48] to identify extracted image segments. By incorporating the depth information, we

also detect the background segment (supporting surface i.e. Table) which is in the form of a planar surface.

Once real objects in the scene are detected, we compute each object pose in signal space. In our work, two pieces of information are required to represent an identified object: position and orientation. The position of each object is computed in Cartesian space. To associate a position to an object, we model the object with a single point located at the center of mass. The orientation of objects is defined as the angle that the main axis of the object makes with respect to the X-axis of the reference frame. Note that, we extract the orientation information only for elongated objects (e.g. cucumber) but not for symmetric objects (e.g. apple). The abstract pose of the respective object is finally estimated from its major axis derived by principle component analysis (PCA). The orientation information is used to find the parameters of the primitives for elongated objects.

Note that the position of the manipulator, i.e. robot end effector, is directly calculated from position sensors and the kinematics of the arm.

2.1.2.2 Real Relations

The *real relations* are the values of relations between pairs of objects in the scene. To detect these values, we use a combination of proprioceptive (e.g. position) and exteroceptive (e.g. tactile, force, and vision) sensors.

When it comes to detecting object relations, there are three phases: before, during, and after the action. In the first and last phases, the only source of information is the vision interface, which essentially computes the Euclidean distance between segmented object point clouds to decide whether they touch each other or not. An example of this detection is shown in Figure 2.4.

While the action is being performed, the data acquired by other sensors (position, force and tactile) are used in addition to the vision system. The data collected from these sensors are fused using several *heuristic* rules, which are conjunctions of individual conditions on different sensor data. For example, the first rule to detect the relation of *manipulator* with *main* object is a combination of conditions on two sensors: position and tactile. This rule declares a touching relation when the Euclidean distance between

TABLE 2.3: Rules for detecting the spatial relational changes during action execution. Note that $\|manip, O_i\|$ represents the Euclidean distance between manipulator end-effector and position of object O_i . Similarly, $\|O_i, O_j\|$ is the distance between objects O_i and O_j . O_i and O_j are any pair of objects in the scene. $contact_z$ means that the sensor sensed a contact in Z axis, while $!contact_z$ means there is no such contact. $grasped$ means the hand has grasped some object, while $!grasped$ means the hand is empty. The parameters \mathcal{D}_{ij} are the distance thresholds to decide whether two objects are touching or not.

Rule	Relation	Change	conditions					$\mathcal{R}(manip, O_i)$
			vision	position	force	tactile		
1	$\mathcal{R}(manip, O_i)$	N to T	-	$\ manip, O_i\ < \mathcal{D}_{11}$	-	$grasped$	-	
2	$\mathcal{R}(manip, O_i)$	N to T	-	$\ manip, O_i\ < \mathcal{D}_{21}$	$contact_z$	-	-	
3	$\mathcal{R}(manip, O_i)$	T to N	-	-	-	$!grasped$	-	
4	$\mathcal{R}(manip, O_i)$	T to N	-	-	$!contact_z$	-	-	
5	$\mathcal{R}(O_i, O_j)$	N to T	$\ O_i, O_j\ < \mathcal{D}_{51}$	$\ manip, O_j\ < \mathcal{D}_{52}$	$contact_z$	$grasped$	T	
6	$\mathcal{R}(O_i, O_j)$	N to T	$\ O_i, O_j\ < \mathcal{D}_{61}$	$\ manip, O_j\ < \mathcal{D}_{62}$	-	$grasped$	T	
7	$\mathcal{R}(O_i, O_j)$	T to N	$\ O_i, O_j\ > \mathcal{D}_{71}$	$\ manip, O_j\ > \mathcal{D}_{72}$	-	$grasped$	T	
8	$\mathcal{R}(O_i, O_j)$	N to T	$\ O_i, O_j\ < \mathcal{D}_{81}$	-	-	-	-	
9	$\mathcal{R}(O_i, O_j)$	T to N	$\ O_i, O_j\ > \mathcal{D}_{91}$	-	-	-	-	

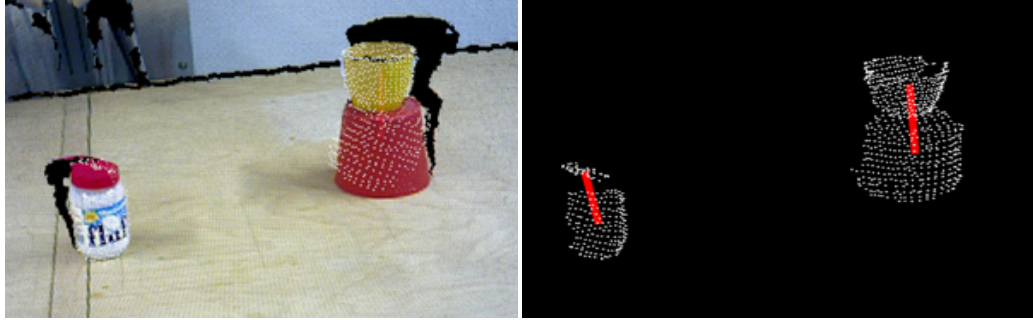


FIGURE 2.4: Calculating the touching relation with our visual perception interface. The red lid is touching the jar and the yellow cup is on top of the bucket (left). The XZY point clouds of the same scene are shown. The red lines indicate a touching relation exists between a pair of objects (right)

the two objects is less than a threshold (denoted by \mathcal{D}_{11}) and the tactile sensor detects a grasp. These rules are listed in Table 2.3.

The rules of Table 2.3 use some intermediate signals which are abstractions of force and tactile sensor data: *contact* and *grasped*. These are flags showing when the robot arm is touching the environment (*contact*) or the robot hand is grasping an object (*grasped*). The *grasped* flag is set to one if the average values of tactile sensors on all three fingers exceed a threshold. An example of tactile sensor readings during an action are shown in Figure 2.5 (top). The solid red line shows the raised *grasped* flag at the times that the hand is grasping some object.

The *contact* flags raise when the external force applied to the end effector of the robot arm exceeds a threshold. In the rules of Table 2.3, we have only used the $contact_z$ flag

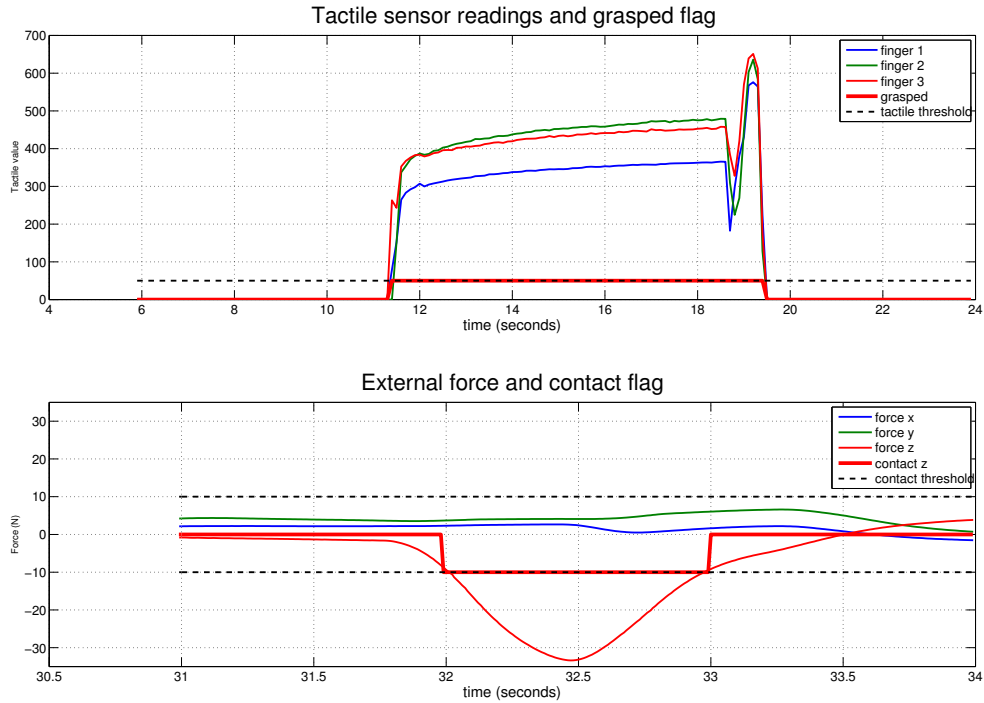


FIGURE 2.5: Intermediate exteroceptive sensory input in the process of analyzing the spatial relation rules given in Table 2.3. The tactile sensor values are used to detect when an object is grasped by the hand (top). The external force signals are processed to detect the touching event (bottom). Here a contact in Z direction is detected.

TABLE 2.4: Rule consideration for detecting spatial object relations (See Table 2.3) and Figure 2.3.

Action \ Relation	\mathcal{R}_1	\mathcal{R}_2	\mathcal{R}_3	\mathcal{R}_4	\mathcal{R}_5	\mathcal{R}_6	\mathcal{R}_7	\mathcal{R}_8
Pick and place	Rule 1,3	-	-	Rule 5	Rule 7	-	-	-
Put on top	Rule 1,3	-	-	Rule 5	Rule 7	Rule 8,9	-	Rule 7
Take Down	Rule 1,3	-	-	Rule 5	Rule 7	Rule 8,9	Rule 5	-
Stir	Rule 1,3	-	-	Rule 6	-	Rule 8,9	-	-
Cut	Rule 1,3	-	-	Rule 5	Rule 6	Rule 8,9	-	-
Poke	Rule 2,4	-	-	-	-	Rule 8,9	-	-
Push with grasp	Rule 1,3	-	-	-	-	Rule 8,9	-	-
Push with holding	Rule 2,4	-	-	-	-	Rule 8,9	-	-
Push apart by holding	Rule 2,4	-	-	-	Rule 9	Rule 8,9	-	-
Push together by holding	Rule 2,4	-	-	Rule 8	-	Rule 8,9	-	-
Pour	Rule 1,3	Rule 8,9	Rule 8,9	Rule 7	Rule 5	Rule 8,9	-	-
Unload	Rule 1,3	Rule 8,9	Rule 8,9	Rule 7	Rule 5	Rule 8,9	-	-

which shows contact along in Cartesian Z axis. Figure 2.5 (bottom) shows an example of contact detected along Z axis.

Since multiple rules exist to detect the same relation in Table 2.3, we should assign, which rules need to be considered in each action. This is summarized in Table 2.4 for the actions in the library.

2.1.2.3 Real Primitives

In Section 2.1.1.4 we defined the abstract primitives. Here, we re-introduce these primitives by adding their parameters and describe their implementations at the low-level.

For the robot arm, we have the following primitives:

- $arm_move(object, T_{off}, \mathcal{P})$
- $arm_move_periodic(a_x, a_y, a_z, b_x, b_y, b_z, \omega)$,
- $arm_rotate(v, \theta)$,
- $arm_exert(F_{des})$,

And for the robot hand we have defined the following primitives:

- $hand_preshape(q)$
- $hand_grasp()$
- $hand_release()$

Here we explain these primitives in more detail and discuss their specific implementation in our system:

arm_move(object, T_{off}, P)

This primitive moves the end effector from the current pose to a pose relative to *object*. The offset of the target from object is stored in homogeneous transformation T_{off} . Equation 2.1 shows how the goal of this primitive is calculated from the pose of the object and the offset transformation.

$$P_{goal} = T_{off} P_{obj} \quad (2.1)$$

The parameters of the trajectory are stored in P . We use Dynamic Movement Primitives (DMP) with joining [49] to generate trajectories. The following dynamic

equations construct trajectory in one dimension from start s toward goal g . Using the terms used in DMP papers the transformation system is defined by:

$$\tau \dot{z} = \alpha_z (\beta_z (r - y) - z) + f \quad (2.2)$$

$$\tau \dot{y} = z \quad (2.3)$$

$$\tau \dot{r} = \begin{cases} \frac{\Delta t}{T} (g - r) & \text{if } t \leq T \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

where α_z , β_z and α_g are time constants and τ is a temporal scaling factor. \dot{z} , \dot{y} and y are acceleration, velocity and position respectively. r is a linear goal function which goes from s to g . The nonlinear function f is used to deflect the trajectory from a straight line. In our application this function is used to shape the trajectories.

$$f = \alpha_\omega \frac{\sum_{i=1}^n \psi_i \omega_i v}{\sum_i \psi_i} (g - y_0) \quad (2.5)$$

The variable v has a dynamic defined by the canonical equation:

$$\tau \dot{v} = - \frac{\alpha_v^s e^{\frac{\alpha_v^s}{\Delta t} (\tau T - t)}}{(1 + e^{\frac{\alpha_v^s}{\Delta t} (\tau T - t)})^2} \quad (2.6)$$

The other parameters of the function f are as follow. The Gaussian kernels ψ_i are defined as:

$$\psi_i = e^{-\left(\frac{t}{\tau T} - c_i\right)^2 / 2\sigma_i^2} \quad (2.7)$$

where c_i , h_i are is the center and width of i th kernel. n is the number of kernels and a set of n weights w_i is used to shape the function f . The parameter α_ω scales all of the weights to adapt to different start and end points. If the weights are learned for a start and goal point s_l and g_l , the scaling factor will be:

$$\alpha_\omega = \frac{|g - s|}{|g_l - s_l|} \quad (2.8)$$

These equations are described in one dimension. To generate trajectories in 3D, we use three of these equations for X , Y and Z axes. An example of the resulting trajectories are shown in Figure 2.6. If we diminish the function f by setting the

weights w_i to zero, we get a straight line from start point to goal point. Examples of straight line trajectories are shown in Figure 2.6 starting from S_1 to S_4 . By using proper values of weights w_i we can generate trajectories with desired shape. For example in Figure 2.6 trajectories which start from S_5 to S_8 have a sine wave shape in Z axis.

So far we showed only the position trajectories. To implement *arm_move* primitive we need to also generate smooth trajectories for orientation, i.e. the rotation of end-effector. Since generating trajectories for Euler angles α, β, γ separately may result in unwanted and unpredictable orientations, we use the “axis/angle“ representation. We know from Euler’s rotation theorem that any rotation in 3D space is equivalent to a single rotation θ_{eq} around a fixed unit vector \mathbf{k}_{eq} . Calculating the new variables θ_{eq} and \mathbf{k}_{eq} from Euler angles, is straight-forward.

The benefit of using axis/angle representing is that we can use a single DMP to generate a trajectory for $\theta(t)$ from zero to θ_{eq} , which results in valid rotations along the path. In addition, since the axis \mathbf{k}_{eq} is by definition fixed throughout one rotation, a single DMP for $\theta(t)$ is enough. For this DMP, we always set the weights to zero.

The generated position and orientation trajectories are fed as desired values to the low-level control system of the robot arm. In our setup we have a KUKA LWR robot which has the following control policy to generate commanded joint torques

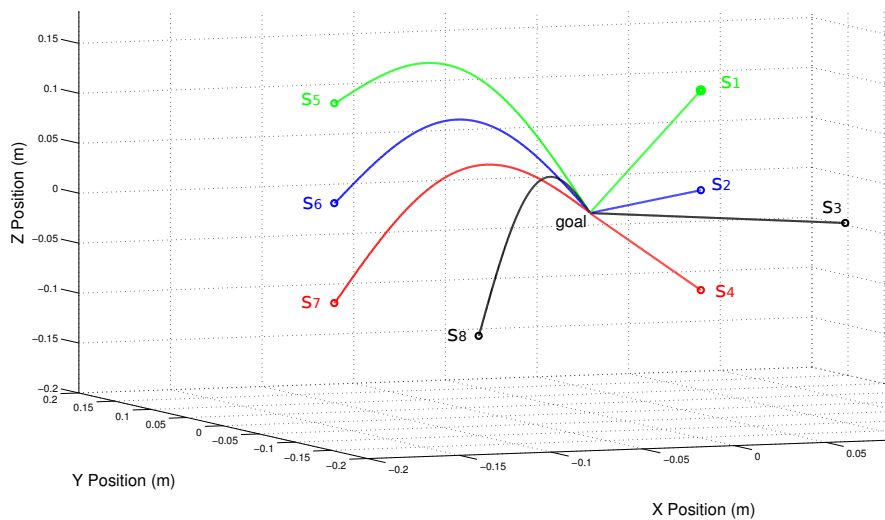


FIGURE 2.6: Example of trajectories generated by DMPs in 3D space. There are different starting points S_1 to S_8 and all end in the origin. The trajectories S_1 to S_4 are straight lines with zero weights while S_5 to S_8 have half-sine shape.

τ_{cmd} :

$$\tau_{cmd} = J^T(k_c(X^* - X)) + D(q) + f_{dyn}(q, \dot{q}, \ddot{q}) \quad (2.9)$$

where X^* is the desired pose, X is the measured actual pose of the robot. The coefficient k_c denotes the gain of the position control which determined the stiffness of the arm during motion. The terms $D(q)$ and $f_{dyn}(q, \dot{q}, \ddot{q})$ are the friction and dynamics of the robot arm which are used in the control system.

***arm_move_periodic*($\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z, \mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z, \omega$)**

For some actions we need to perform some simple periodic motions. There are comprehensive frameworks to create periodic (rhythmic) motions on robots such as rhythmic DMPs. However in our system we only need simple back-and-forth and circular motions, for which a combination of sine and cosine functions suffice. Therefore we implement *arm_move_periodic* primitive using the following equations:

$$x(t) = x(0) + a_x \sin(\omega t) + b_x \cos(\omega t) - b_x \quad (2.10)$$

$$y(t) = y(0) + a_y \sin(\omega t) + b_y \cos(\omega t) - b_y \quad (2.11)$$

$$z(t) = z(0) + a_z \sin(\omega t) + b_z \cos(\omega t) - b_z \quad (2.12)$$

These equations generate smooth trajectories from initial position (which is $[x(0), y(0), z(0)]$). The a_x and b_x determine the strength of sine and cosine components on the X axis. The period of the motion is determined by the parameter ω . Examples of periodic motion generated in actions are shown in Figure 2.7 The back-and-forth motion happens in a cutting action and the circular motion in a stirring.

***arm_rotate*(v, θ)**

This primitive is implemented as a special case of the *arm_move* primitive, using only the rotation part. It implements a pure rotation using the axis/angle convention: A rotation around the axis v with the angle θ . This primitive is used in actions like unscrew, rotate and align, where a pure rotation along a given axis is necessary.

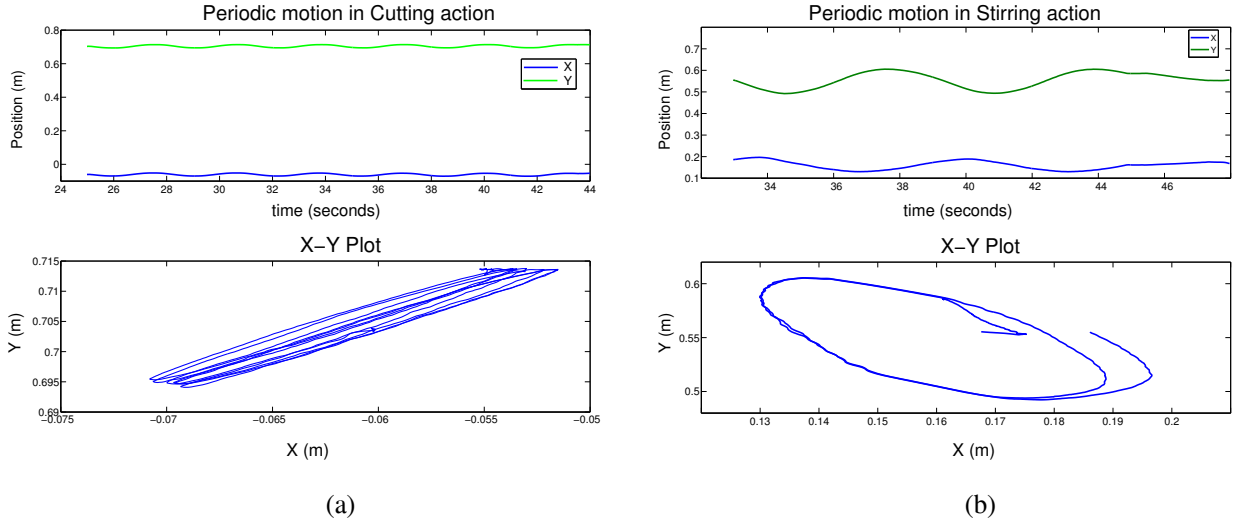


FIGURE 2.7: Examples of periodic trajectories in actions. (a) In the cutting action the following parameters are used to generate a back and forth motion: $a_x = 0. - 008 m$, $a_y = -0.01 m$, $\omega = 1.8 rad/s$. The initial position is $x(0) = -0.6 m$ and $y(0) = 0.7 m$. (b) In the stirring action a circular motion is generated by using the following parameters: $b_x = 0.03 m$, $a_y = -0.05 m$, $\omega = 1 rad/s$. The initial position is $x(0) = 0.187 m$ and $y(0) = 0.55 m$.

arm_exert(F_{des})

Manipulation actions sometimes need more than just pure position control. In some tasks we need to also regulate the force exerted at the environment. Many times it is important to have both position and force control at the same time. For example in a cutting action, the robot arm keeps a force between the knife and banana in the Z direction (constrained space), while moving the knife back and forth in the XY -plane (unconstrained space).

This is possible by using parallel position-force control schemes such as one introduced in [50]. The following control policy is used in this case:

$$\tau_{cmd} = J^T(k_c(X^* - X) + F_{cmd}) + D(q) + f_{dyn}(q, \dot{q}, \ddot{q}) \quad (2.13)$$

Here the term F_{cmd} implements a force control which has feed-forward and PI terms:

$$F_{cmd} = F_{des} + K_p(F_{des} - f) + K_I * \int (F_{des} - f) \quad (2.14)$$

Example of force control is shown in Figure 2.8 where the desired force is $F_{des} = [0, 0, 1]^T N$.

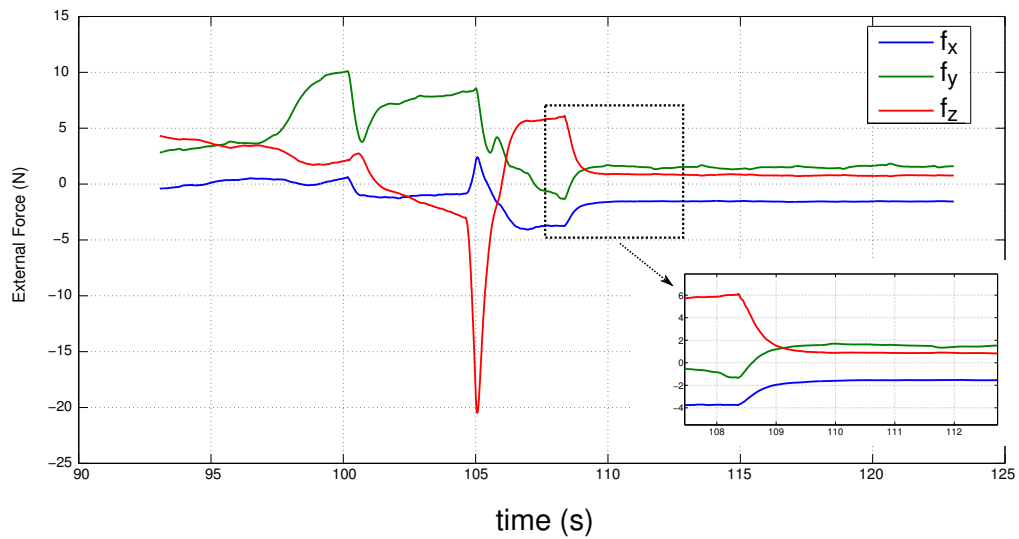


FIGURE 2.8: Example of force control using the *arm_exert* primitive. The desired force is 1 N in Z direction.

hand_preshape(q)

This primitive is used to create a desired shape of the robotic hand. Our robot hand (Schunk SDH-2) has 3 fingers and in total 7 DOFs. The control system of the hand has the ability to move to a desired configuration:

$$q = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]^T \quad (2.15)$$

Two sample configurations are shown in Figure 2.9 which are the power and precision grasps used for round and elongated objects, respectively.

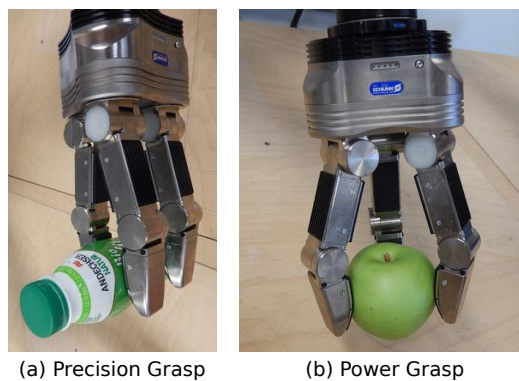


FIGURE 2.9: Two pre-shape configurations are used in our system. The power grasp (right) is used for symmetric objects while the precision grasp (left) is for elongated objects.

hand_grasp()

To manipulate objects usually it is needed to grasp them first. For grasping, we use velocity control of finger joints together with feedback from tactile sensors on the fingers. The combination of *hand_preshape* and *hand_grasp* primitives enables us to grasp simple objects, which is enough to demonstrate the functionality of the proposed execution system. The complex problem of grasping arbitrary objects is not in the scope of this research.

hand_release()

This primitive is used to release a previously grasped object which is simply opening the hand until the tactile sensors show that the object is released.

2.2 Ontology of Actions

In Section 2.1 we listed the necessary components to define and execute manipulation actions. Our goal in this section is to enumerate the actions within this framework and categorize them. We use the object categories defined in Table 2.1 to derive all possible manipulation actions and create an *action ontology*. The actions have three components:

1. Objects
2. SEC matrix (Relations)
3. Primitives

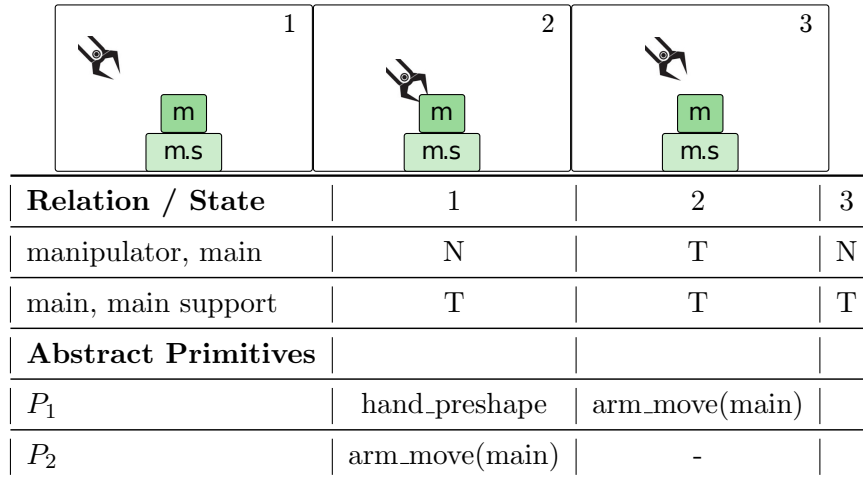
We try to enumerate all possible actions by starting from the simplest actions (involving minimum required objects) and increase the complexity by adding more objects. From constraints 1 and 2 derived in Section 2.1.1, we know that each action at least has the *manipulator* and *main* objects. We also assume that the main object is not floating in space at the beginning (and the end) of an action. It lies on some other object i.e. *main support* or *primary*.

If the *main* object rests on another object throughout the action, the other object is called *main support*. If otherwise there is a T to N transition between the main and its support at the beginning, this object is called *primary* object. These two cases create the first two categories. Another category arises when there is additionally a *load* object. This is an object which is manipulated indirectly by the manipulator.

Category	Sub-Category	Example Actions
Actions with main support	Actions with hand, main and main support	push, punch, flick
	Actions with hand, main, main support and primary	push apart, cut, chop
	Actions with hand, main, main support and secondary	push together
	Actions with hand, main, main support, primary and secondary	push from a to b
Actions without main support.	primary \neq secondary and primary support \neq secondary support	pick and place, break off
	primary \neq secondary and primary support = secondary support	pick and place, break off
	primary \neq secondary and primary = secondary support	put on top
	primary \neq secondary and primary support = secondary	pick apart
Actions with load and container	primary = secondary	pick and place, break off
	The relation of load and main changes from N to T (loading)	Pipetting
	The relation of load and main changes from T to N (unloading)	Pour, Drop

TABLE 2.5: Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category.

The categories are summarized in Table 2.5. The actions in the ontology are listed in Appendix A. Here we briefly explain these categories:

FIGURE 2.10: The action *Poke* is an example from Category 1

Actions with main support

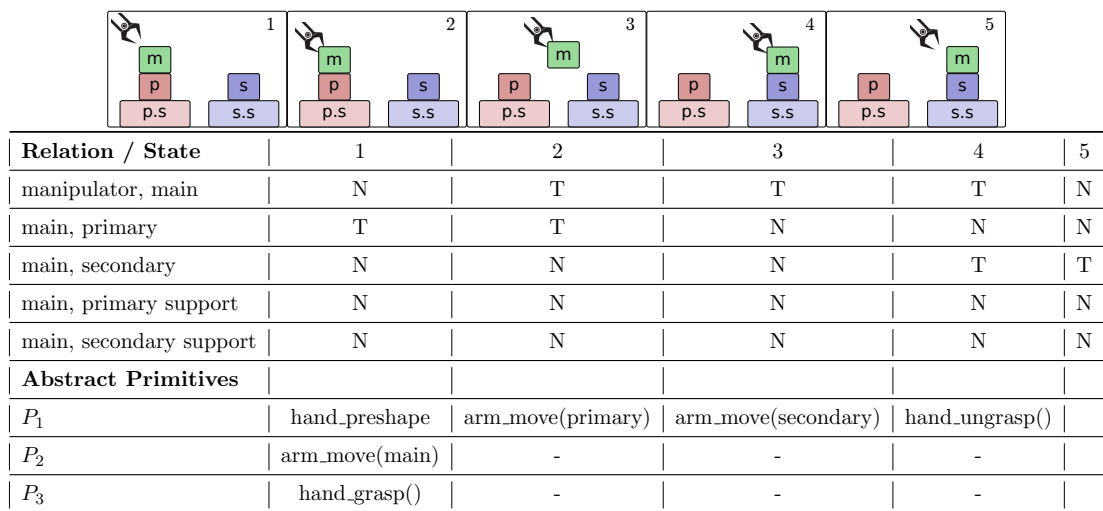
Actions with *manipulator*, *main* and *main support*. During the action, the main object is always attached to the main support. The manipulator approaches the *main* object, touches it and manipulates it. Then it retracts and the action ends. Actions like push, rotate and cut are examples of this category. The structure of a *Poke* action is shown in Figure 2.10.

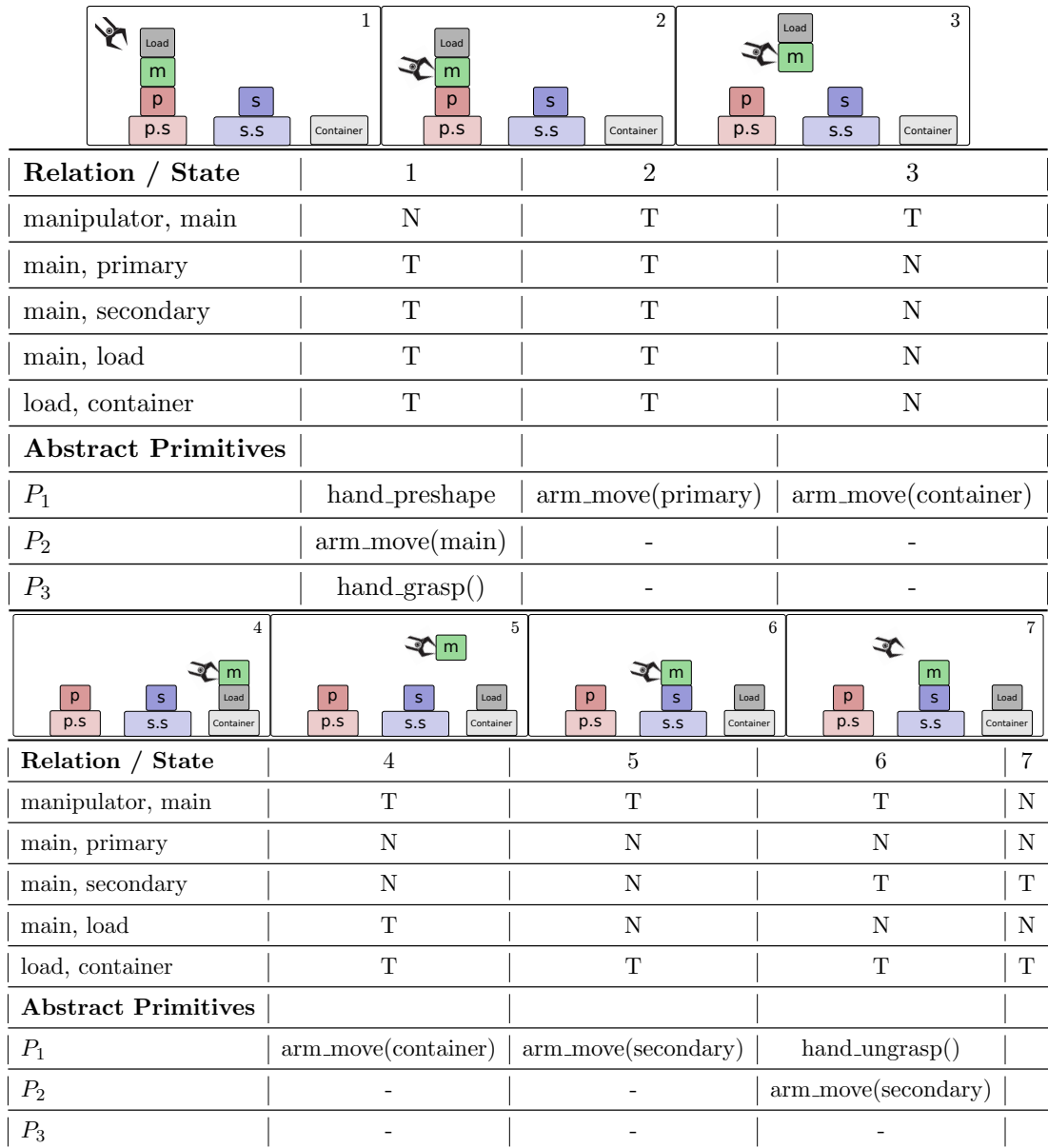
Actions without main support

In the actions of this category, there is no *main support*. This usually means that there is an *air phase* in these actions, in which the manipulator grasps the main and lifts it. In these actions we can always identify a *primary* and a *secondary* object (although sometimes they are the same object). Actions like *Pick and place*, *Put on top* are in this category. The structure of a *Pick and place* action is shown in Figure 2.11.

Actions with load

Actions of this category include an object which is manipulated or transferred indirectly (*load* object). Here we have actions like load, unload and pour. The structure of an *Unload* action is shown in Figure 2.12.

FIGURE 2.11: The action *Pick and place* is an example from Category 2

FIGURE 2.12: The action *Unload* is an example from Category 3

2.3 Action Execution

In this section the details of action execution will be presented which relies on the action definition presented in Section 2.1. The definition has symbolic and sub-symbolic components which need to be connected in the execution phase. To this end we introduce a mid-level whose main component is a finite state machine (FSM). We define the states, inputs, outputs and transitions of the FSM in a way that it serves as a mid-level in the action definition and enables us to execute them. Afterwards, we introduce an action execution engine which is a state diagram implementing the execution in a robotic software framework.

2.3.1 Mid-level sequencer: Finite State Machine

In order to execute the defined actions, the primitives need to be executed in a sequence, taking into account the relations between objects. There should be a connection between the high- and low-level components. This connection is established by introducing a FSM as the middle layer of actions.

An FSM is a logic unit, that determines outputs and the next state of a system, based on the current state and inputs. It is formally defined as a 6-tuple (S, U, Y, f_1, f_2, s_1) . The following paragraphs show the definition of these variables. It is also shown how these variables are related to the high-level and low-level components. In particular, the variables for a *Put on top* action are derived to show a complete example. For the *Put on top* example the resulting FSM is depicted in Figure 2.13.

2.3.1.1 States(S)

There exists a finite set of states in an FSM, denoted by S_i . The interpretation of these states depends on the application. In our application, the touching relation of objects determines the state. In execution, the number of states is equal to the number of columns of the SEC matrix. For example, the SEC matrix of *Put on top* has five columns, therefore the FSM has five states : $S = \{S(1), S(2), \dots, S(5)\}$ as shown in Figure 2.13.

2.3.1.2 Inputs(U)

The inputs of the FSM, should be mapped to the inputs of the system. The inputs help to decide which state transition should be made. In action execution, the inputs are the real relations of the object in the scene. The number of inputs depends on the number of objects involved in the action. In our example, there are 5 abstract relations, therefore we need 5 inputs $U = [U(1), U(2), U(3), U(4), U(5)]^T$ where:

$$U(1) = \mathcal{R}(\text{manipulator}, \text{main})$$

$$U(2) = \mathcal{R}(\text{main}, \text{secondary})$$

$$U(3) = \mathcal{R}(\text{main}, \text{primary})$$

$$U(4) = \mathcal{R}(\text{primary}, \text{secondary})$$

$$U(5) = \mathcal{R}(\text{main}, \text{secondary sup.})$$

The inputs U are shown near the transition arrows in Figure 2.13.

2.3.1.3 Outputs(Y)

An FSM can produce outputs depending on current state and inputs. These outputs are defined in terms of defined commands of the application. In our application, the outputs are a set of arm/hand primitives, which should be sent to the control system. In the *Put on top* example there are five states, therefore there are five outputs $Y = \{Y(1), Y(2) \dots, Y(5)\}$, shown in Figure 2.13:

$$Y(1) = \left[\text{hand_preshape}() \quad \text{arm_move}(\text{main}) \quad \text{hand_grasp}() \right]$$

$$Y(2) = \left[\text{arm_move}(\text{primary}) \quad - \quad - \right]$$

$$Y(3) = \left[\text{arm_move}(\text{secondary}) \quad - \quad - \right]$$

$$Y(4) = \left[\text{hand_release}() \quad \text{arm_move}(\text{secondary}) \quad - \right]$$

$$Y(5) = \left[\quad \quad \right]$$

Since the last state is already the goal state, the last output is always empty.

2.3.1.4 State transition function(f_1)

The state transition function determines the next state, based on current state and inputs.

In action execution, the transition from state i to $i + 1$ happens when the inputs are equal to the $(i + 1)$ th column of the SEC matrix. When checking the equality, the inputs that correspond to don't-care relations are not taken into account. Hence, the transition function for *Put on top* example is as follows:

$$f_1(S(i), U) = \begin{cases} S(i + 1) & \text{if } U = SEC(:, i + 1) \\ S(i) & \text{otherwise} \end{cases} \quad (2.16)$$

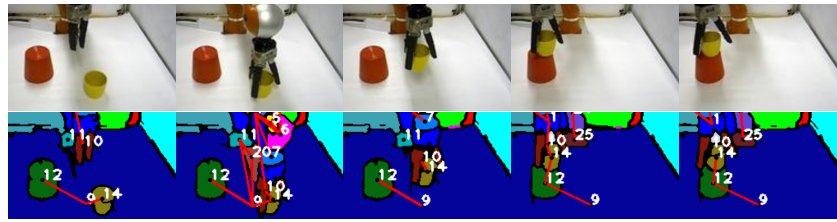
2.3.1.5 Output function (f_2)

In general, the outputs of an FSM depend on state and inputs. However, here we use a specific kind of FSM called Moore machine, in which outputs depend only on the states. Hence, we have a simple output function which associates outputs to each state:

$$f_2(S(i)) = Y(i) \quad (2.17)$$

2.3.1.6 Initial state($S(1)$)

The initial state is associated with the first columns of the SEC that represents the initial relations between objects. It is assumed that at the beginning the real relations match the first column of the SEC matrix, otherwise the action won't start.



Relation/State	S_1	S_2	S_3	S_4	S_5
R (manipulator, main)	N	T	T	T	N
R (manipulator, secondary)	A	A	A	A	A
R (manipulator, primary)	A	A	A	A	A
R (main, secondary)	N	N	N	T	T
R (main, primary)	T	T	N	N	N
R (secondary, primary)	T	T	T	T	T
R (main, primary sup.)	A	A	A	A	A
R (main, secondary sup.)	T	T	N	N	N
Primitive 1	hand _preshape	arm_move (prim.)	arm_move (sec.)	hand _release	-
Primitive 2	arm_move (main)	-	-	arm_move (sec.)	-
Primitive 3	hand_grasp	-	-	-	-
	$Y(1)$	$Y(2)$	$Y(3)$	$Y(4)$	$Y(5)$

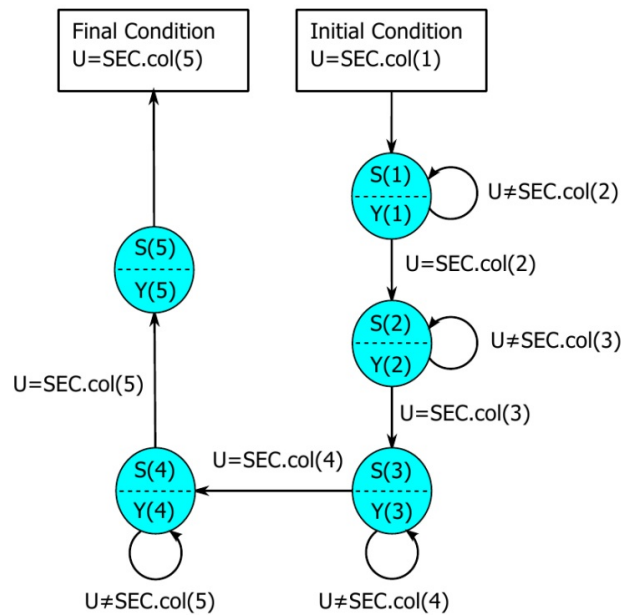


FIGURE 2.13: The finite state machine (FSM) for *Put on top* action is shown. The states of the FSM (S_1 to S_5) are related to the columns of SEC matrix and states of the action. The action starts if initially inputs(real relations) match the first column of the SEC matrix. At each state, the FSM outputs (Y_1 to Y_5) are executed which are the defined primitives of each column. The transition to from state i to $i + 1$ occurs when the inputs match the $i + 1$ -th column of the SEC matrix. Otherwise, the next state is the same as the current state (loop transitions). This process is continued until reaching the final state which corresponds to the last column of the SEC matrix.

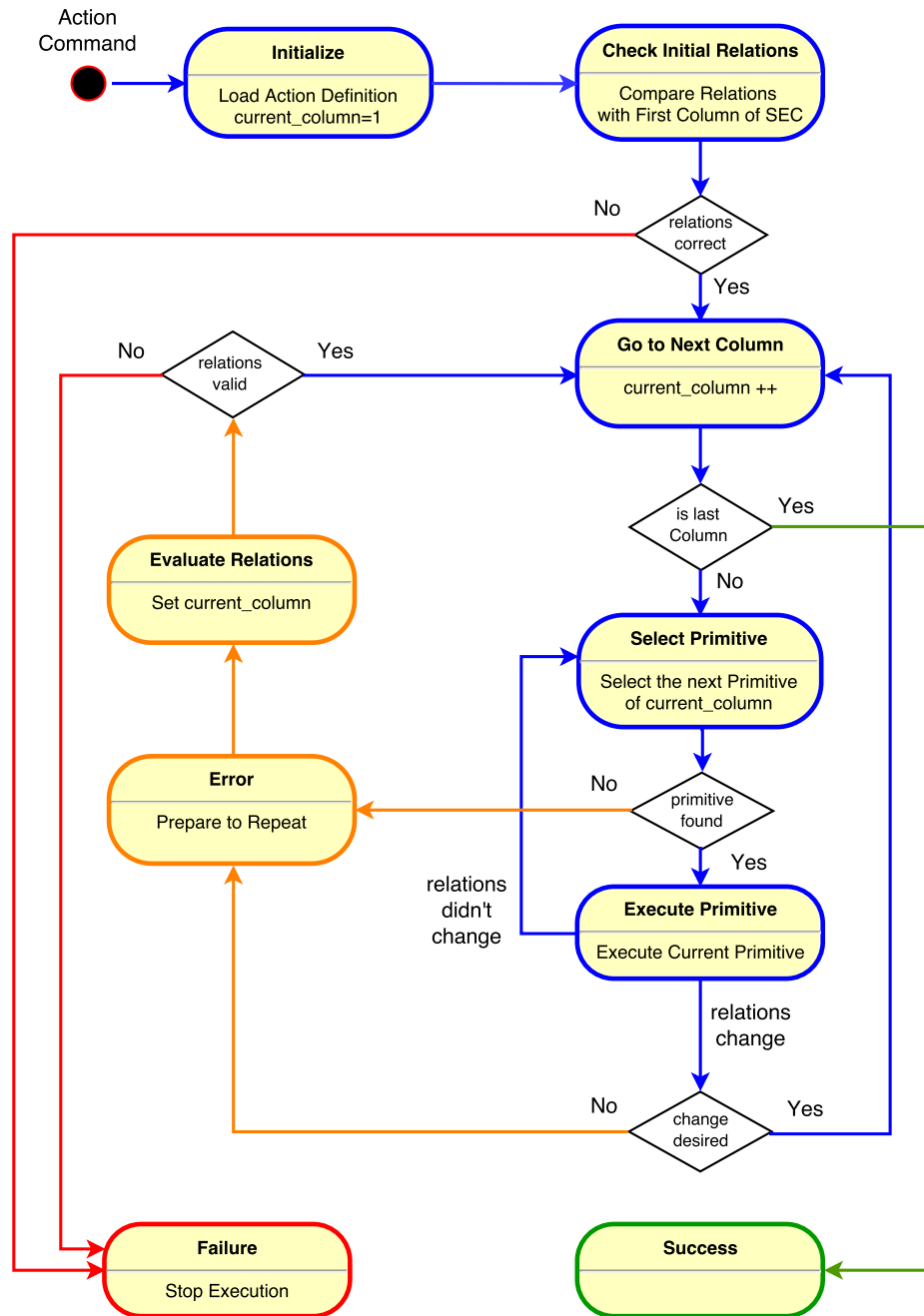


FIGURE 2.14: State diagram of the execution engine which controls the execution of actions. This state machine is the main component of the mid-level. The diagram is shown more clearly, with different colors used for normal execution (blue), error handling (orange), failure (red) and success (green) states and transitions.

2.3.2 Action Execution Engine

The finite state machine described in Section 2.3.1 must be implemented in a software which is able to communicate with sensors and other robot hardware. We developed an action execution engine as a component of the OROCOS (Open Robot Control Software)

framework ([51, 52]). The OROCOS framework provides tools to develop real-time robotic software.

The overview of the execution process is shown in Figure 2.14. The execution engine starts by receiving an action command and after checking the initial relation, moves forward and executes the primitives one by one in order to create the desired changes in relations. The variable *current_column* is used to track the progress of the action and points to the current column of SEC matrix. The details of states and transitions are as follows:

- **Initialize:** After receiving a new action command, the high-level definition of the desired action is loaded. The variable *current_column* is set to 1 to point to the first column of SEC matrix. The action command consists of the action type, the main and other objects involved in the action.
- **Check Initial Relations:** Here the current relations of objects are compared to the first column of SEC matrix of the commanded action. The pre-condition of executing the action is that the two are equal, otherwise the FSM transitions to the *Failure* state.
- **Go to Next Column:** This state increments the variable *current_column* and causes the action to progress from one column of SEC to the next. If we are already in the last column of SEC, it transitions to *Success* state, which means the action is done successfully.
- **Select Primitive:** In this state the next primitive of the current SEC column is selected. If available, we transition to the *Execute Primitive* state. Otherwise we are entering the *Error* state and we need to handle the error, since this implies that all primitives of the current SEC column are executed but the desired changes in relations did not happen.
- **Execute Primitive:** The selected primitive is executed here. This state has several sub-states, each performing one type of primitive. To keep the diagram simple, they are not shown in Figure 2.14. In this state, the relations of objects are monitored and if they change to the desired values, we transition to the *Go to Next Column* state. If relations change to unwanted values, the next state

will be the *Error* state. Finally if the primitive is done and no change in relations is detected, it transitions to the *Select Primitive* state, to look for the next primitive.

- **Error:** This state indicates that the execution of the current action is not progressing as expected. However, there is still hope to recover from the error, and continue the execution. There are two ways to enter this state. First, the primitives defined for the current SEC column are all executed but the desired change in relations has not occurred. Second, during the execution of primitives an unwanted change in relations happened.

In this state we try to go back to a previously known state of the action, and continue from that point. Usually this means that the robot arm retracts from the scene and receives new object poses and relations. After receiving the new perception, we transition to *Evaluate Relations* state. In Section 4.1 we will show some examples of handling errors.

- **Evaluate Relations:** After receiving the new perception in *Error* state, we evaluate the current situation of the objects in this state. If for these relations, we could continue from the last known state, we transition to *Go to Next Column* state and continue the execution. Otherwise, we go to *Failure* state since we are in an invalid state and can not proceed.
- **Failure:** If the relations of objects are in a way that there is no known way to proceed the execution, we transition to *Failure* state. At this point we stop the execution and announce failure. The failure is reported to the operator or the high-level planner, so that a proper decision can be made. Note that there is no high-level planner introduced here, since it is not in the scope of this work.
- **Success:** This state is entered if the action is successfully executed according to the SEC matrix.

2.4 Results

In this section, we will present various experimental results of our proposed action execution framework. Results cover execution of both, single actions (e.g. *Cutting, Pushing*, etc.) and chained actions such as “*making a salad*”. Before presenting these results, we briefly introduce our hardware and software tools used in the experiments.

2.4.1 Hardware

Our setup consists of a robot manipulator, a three-finger robotic hand and a vision interface.

2.4.1.1 Robot Arm

Our robot arm is a KUKA LWR (Light Weight Robot) IV manipulator. It is a kinematically redundant anthropomorphic manipulator developed jointly by KUKA Robot Group and the German Aerospace Center (DLR). It has 7 DOFs and is equipped with position and torque sensors at each joint. It estimates the external torques applied to each joint which also gives an estimate of external force and torque at the end-effector. The robot can be controlled both in joint and Cartesian spaces with variable compliance and damping.

2.4.1.2 Robot Hand

Our robot hand is a Schunk Dexterous Hand 2 (SDH-2) produced by the company Schunk. It has three fingers and 7 degrees of freedom, which can be controlled in position or velocity modes. It is equipped with two tactile sensors on each finger, that provide feedback while grasping objects.

2.4.1.3 Vision System

Our vision system includes a static RGB-D (Asus Xtion) sensor and a DSLR camera (Nikon D7200). The RGB-D sensor provides both color and depth cues which are processed for image segmentation and tracking issues. The DSLR camera is further

integrated into the vision system to capture high resolution images of the scene for the purpose of object recognition. The vision system is developed using the ROS framework (See [47] and [48]).

2.4.2 Single Actions

To quantitatively evaluate the proposed action execution framework, we conducted a large set of experiments with several types of actions and objects. The central goal here is to benchmark the success of the execution of actions provided in the library. Note, to arrive at a useful characterization of this framework all actions are analyzed *without error handling*. Only by this decisive percent-success values can be measured.

We are not concerned with complex computer vision, thus, colored and texture-less objects were mostly preferred in the experiments to cope with the intrinsic limitations of the imaging sensors and to have more reliable visual segmentation of perceived scenes. Figure 2.15 illustrates the set of manipulated objects, which contains in total 19 different objects in 8 categories:

- | | |
|----------------|-----------|
| 1. round fruit | 5. plates |
| 2. long fruit | 6. knives |
| 3. cups | 7. spoons |
| 4. cubes | 8. boards |



FIGURE 2.15: The set of objects used in our experiments. There are in total 19 objects in 8 categories: 1-Round fruits 2-Long fruits 3- Cubes 4-Cups 5-Containers 6-Plates 7-Spoons 8-Knives.

TABLE 2.6: List of 10 atomic actions stored in the library and also used in experiments introduced in Section.2.4.2. The last two columns show sample objects used in each action.

#	Action Name	Explanation	Main	Tool	Primary	Secondary
1	Pick and Place	The main object is picked from primary and placed on the same object.	cup, apple, orange, cucumber, eggplant, bucket, plate, box	- - -	table	table
2	Push with Grasp	The main object is pushed to the goal position after being grasped.	box, apple, orange cucumber	- -	- -	- -
3	Push with Holding	The main object is pushed to the goal position after being held on top	box, apple, orange	-	-	
4	Poke	The main object is poked.	box, apple, orange	-	-	
5	Put on Top	The main object is put on top of the secondary object.	cup, cucumber, apple, orange	- -	table table	box, bucket, cup plate, board
6	Take Down	The main object is taken down from the primary object.	cup, banana, apple, orange	- -	box, bucket, cup board	table table
7	Push apart by holding	The main object is pushed apart from primary after being held from top.	orange, box	-	apple, cup	
8	Push together by holding	The main object is pushed to secondary after being held from top.	apple, orange	-		box
9	Cutting	The main object is cut by the tool.	zucchini, cucumber banana	knife		
10	Stirring	The main object is stirred by the tool object.	bucket	spoon, knife		

The first ten actions defined in Table 2.4 are performed using objects of various types, sizes, shapes and poses (see Figure 2.15). The executed single actions with their brief explanations and involved objects are listed in Table 2.6.

We evaluate the results of experiments in two ways. First, the success rate of execution of each action type is measured. These results give an overview of the execution performance of actions on different object categories presented in various scene contexts. Thus, we can measure the robustness as well as the generalization capacity of the proposed action library. Second, we plot variations in the low-level sensory input, such as tactile, position, and contact signals, while the action is being executed. In these results, we can obtain information on the underlying perception mechanism in the execution framework and the discretization of the low-level continuous sensory data to reach high-level symbolic action representation.

We have selected 10 actions from the ontology and applied them to 19 objects in 8 categories. To evaluate success rates, we repeated 10 single actions on 10 different object sets and repeated each with 3 different poses. Thus, we obtained 30 trials for

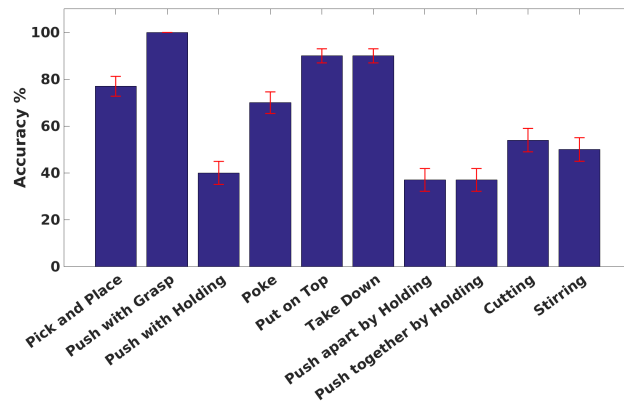


FIGURE 2.16: Overall success rate of 10 atomic action execution after 30 trials for each.

each action, i.e. in total 300 experiments. The overall success rate per action type is shown in Figure 2.16. Red bars in the figure depict the standard error of the mean.

The first result is that in 7 out of 10 actions, the success rate is equal or more that 50 percent. This shows that the system is able to cope with different objects and poses

Action Name	No. of Trials	Grasp type	Main object category						Tool		Primary/ Secondary object category								Success		Average Accuracy (%)	
			Power	Round fruit	Long fruit	cup	container	cube	plate	knife	spoon	Round fruit	Long fruit	cup	container	cube	plate	knife	spoon	ratio		%
				Precise	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			
Pick and Place	30	Power	6/6	-	6/6	0/3	2/3	0/3	-	-	-	-	-	-	-	-	-	-	14/21	67		
			Precise	9	-	6/6	-	3/3	-	-	-	-	-	-	-	-	-	-	-	9/9	100	
Push with Grasp	30	Power	9/9	-	6/6	-	3/3	-	-	-	-	-	-	-	-	-	-	-	18/18	100		
			Precise	12	-	9/9	-	3/3	-	-	-	-	-	-	-	-	-	-	-	12/12	100	
Push with Holding	30	-	4/6	2/9	0/6	-	6/6	0/3	-	-	-	-	-	-	-	-	-	12/30	40			
Poke	30	-	6/6	6/6	0/6	-	6/6	3/6	-	-	-	-	-	-	-	-	-	21/30	70			
Put on top	30	Power	9/9	-	6/6	-	3/6	-	-	-	-	-	3/3	6/6	3/3	15/18	-	-	18/21	85		
			Precise	9	-	9/9	-	-	-	-	-	-	-	-	-	-	-	-	-	9/9	100	
Take down	30	Power	6/9	-	9/9	-	3/3	-	-	-	-	-	6/6	6/6	6/6	9/12	-	-	18/21	85		
			Precise	9	-	9/9	-	-	-	-	-	-	-	-	-	-	-	-	-	9/9	100	
Push apart by holding	30	-	3/6	2/9	0/6	-	6/6	0/3	-	-	6/6	-	-	-	5/24	-	-	11/30	37			
Push together by holding	30	-	3/6	2/9	0/6	-	6/6	0/3	-	-	6/6	-	-	-	5/24	-	-	11/30	37			
Cutting	30	-	0/6	16/24	-	-	-	-	-	16/30	-	-	-	-	-	-	-	16/30	54			
Stirring	30	-	3/12	12/12	-	0/6	-	-	-	15/30	-	-	-	-	-	-	-	15/30	50			

FIGURE 2.17: Success rate of executing actions in each object category. Each action is executed 30 times using different object sets. The ratio of successful trials are shown for each object category (middle columns). For actions involving grasp, the results are separately shown for each grasp type. The overall success rates on each grasp type and average success scores are shown in the last two columns. The values in the last column match the final average accuracy rates shown in Figure 2.16.

for most of the actions. The second impression that the figure conveys is that there is a prevalent failure, mostly observed, in the execution of pushing actions which were mainly performed by just holding objects without applying any certain grasp, e.g. *push with holding* described in Table 2.6. The overall accuracy was measured as 64.5% and this value reached 75.8% in the case of excluding those failed pushing types. The main reason of this accuracy drop in pushing actions is due to the shape of manipulated objects. For instance, while the robot was gently holding the object, e.g. an apple, to push it, the object slipped over the contact surface and, thus, led to a failure of the action. It is known that such types of actions are exceedingly difficult for robots but also for humans and we have often to reactively correct grasp and push to succeed. Hence, building in reactive correction mechanisms would certainly mitigate this problem.

We also observed a low success rate of about 50% for the cutting and stirring action types. In the stirring action, some of the failures occurred because the manipulated spoons were slightly too big for the containers. In the case of the cutting action, failures were due to inability to cut thick objects such as round fruits (apple or orange). Human cutting operations are heavily dominated by reacting to the “feel” of cutting and correcting force and angle.

A more detailed analysis on the execution of single actions is given in Figure 2.17. The results are separately computed for each individual action and object category. For those actions which require object grasping, the results are also categorized according to the grasp type. We here note that object grasping is not in the focus of this study and therefore in our experiments we only considered two types of grasps: power and precision. The average success rate for each grasp type and for the entire experiment are shown in the last three columns. For instance, in *Pick and place* action, 21 out of 30 trials were successfully performed with power grasp, which led to 67% average accuracy, whereas it was computed as 100% for the precision grasp. The details of single action experiments are provided in Appendix B.

We can further analyze the cause of failures. We divide failures into two groups: systematic and random. A systematic failure is the one that happens every time when we perform a certain action on a certain object category. In other words, a systematic failure is the indication of applying an action on a wrong object category. For example, cutting cups with a knife or trying to stir with round fruits. On the other hand, a

random failure may or may not occur depending on the pose or size of the manipulated objects. This type of failure may be prevented by modifying the object size or pose. For example, we could not stir with a big spoon in a small cup, but replacing with a smaller spoon, stirring actions in the same cup succeeded.

In this regard, for the entries in Fig.2.17 indicating that the action does not succeed regardless of the object sizes and poses, i.e. due to a systematic failure, we conclude that the selected object category does not afford that action. In this sense, the proposed framework can be used to explore action related affordances on different object types, which can further be passed to any high-level task planner. Note that such reasoning also plays a vital role for cognitive robots exploring grounded object affordances.

Next, we take a closer look at the low-level sensory data including the position, tactile, and force contact signals of the robot arm during the experiments. Here, we aim at topological changes in the perceived scene by fusing data from several sensors, and to calculate object relations.

Figure 2.18 shows the position, tactile, and force sensor data together with detected relational changes between objects and the robot arm during the execution of a sample *Put on top* action. The first plot in Figure 2.18 confirms that due to the use of DMPs with joining, the robot arm seamlessly follows the desired goal positions which are indicated with circles. In the second plot in Figure 2.18 we also see that the tactile sensor is activated once the primary object is grasped. In a similar manner, the force sensor reports a contact when the object is placed. All these sensory data together with the visual feedback are fused to detect final spatial relational changes in the scene as described in section 2.1.2.2. Figure 2.18 at the bottom illustrates the extracted SEC representation over time for the *Put on top* action as a colored matrix which is identical to the one stored in the action library as shown in Figure 2.3. This plot confirms that the proposed action execution framework can successfully process continuous sensory data and extract descriptive states in the scene, which yields compact high-level action representation, i.e. SEC.

In Figure 2.19, we show similar plots for a *Cutting* action in which the robot first grasps a knife and then cuts a cucumber into pieces. The position plot on the top highlights the oscillatory motion pattern of the robot arm during the actual cutting phase. Note that in some cases the actual robot position does not meet the goal position. This is expected,

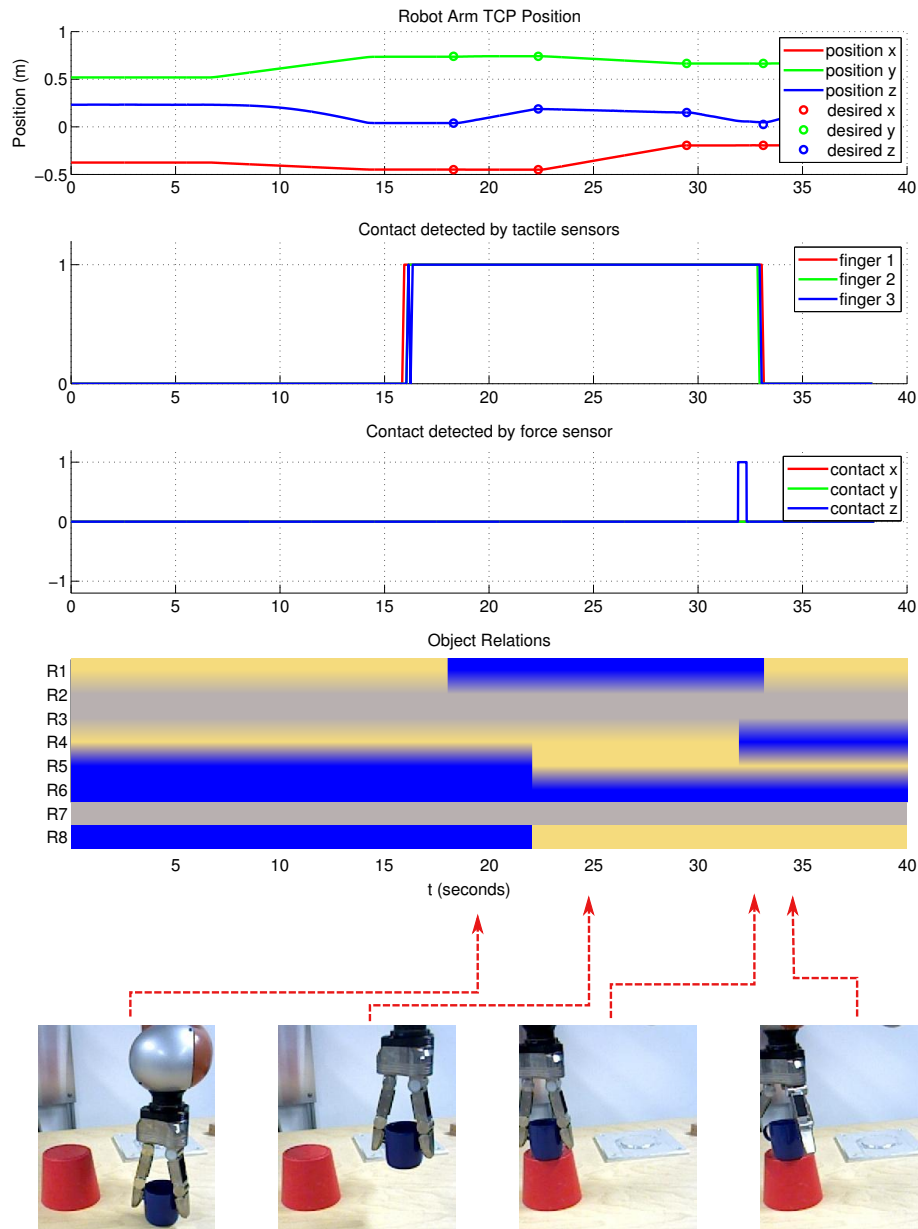


FIGURE 2.18: Low-level sensory data in a sample *put on top* action. The position, tactile and force contact signals are shown on the top. All changes in object contact relations are shown in the bottom plot as a color coded SEC matrix. Here, blue and yellow represent *Touching (T)* and *Not touching (N)* respectively. The gray color shows either *Absence (A)* or relations which are not important (dont-care). Some sample snapshots at the bottom show the scene topology at each state of the action.

since whenever the desired relation changes happen, the current primitive is ended and the state machine moves to the next primitive. The second row shows how the tactile sensors detect the contact which happens at the *hand_grasp()* primitive (grasping the knife) and how this signal vanishes right after the *hand_release()* primitive.

The force signal in Z direction is used to verify the contact between the *tool* and *main*

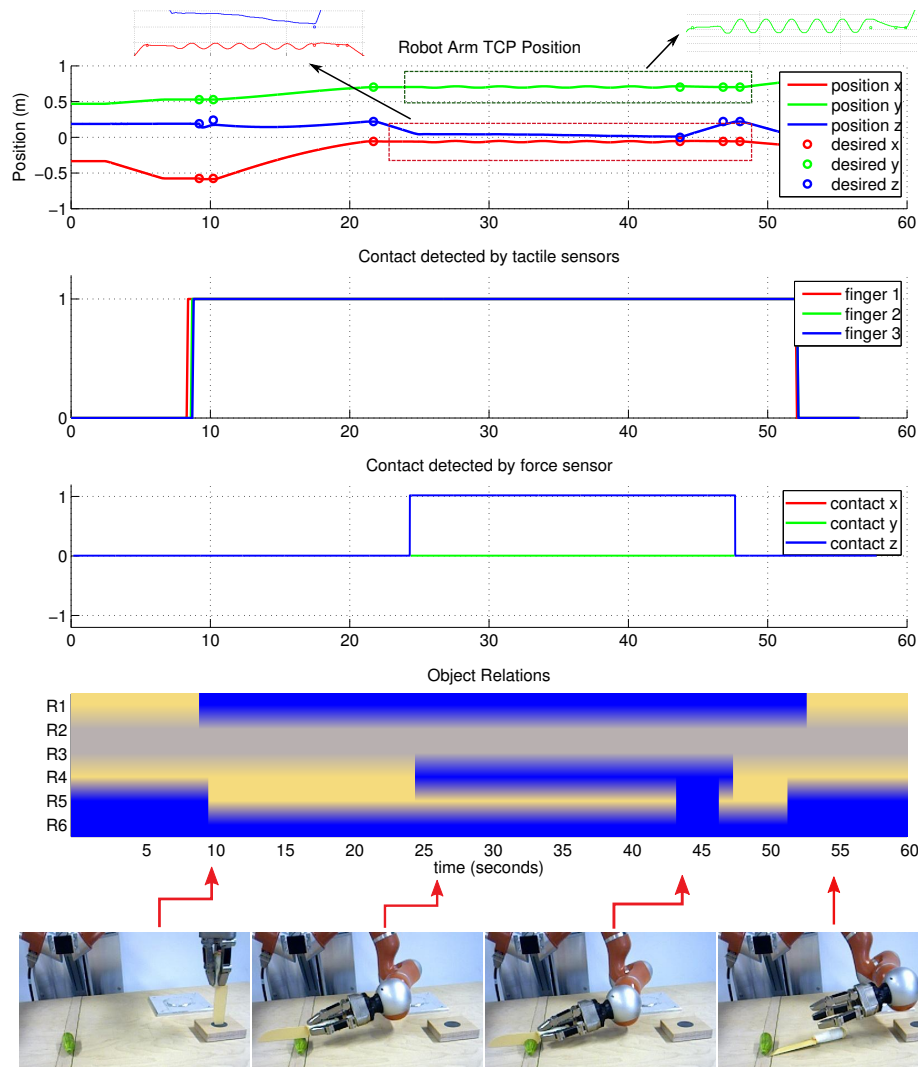


FIGURE 2.19: Low-level sensory data in a sample *cutting* action. The position, tactile and force contact signals are shown on the top. In the cutting action, a part of the trajectory corresponding to the back and forth motion of knife is zoomed in to show the oscillatory motion pattern. All changes in object contact relations are shown in the bottom plot as a color coded SEC matrix. Here, blue and yellow represent *Touching* (T) and *Not touching* (N) respectively. The gray color shows either *Absence* (A) or relations which are not important (dont-care). Some sample snapshots at the bottom show the scene topology at each state of the action.

objects, in this case the knife and cucumber, which triggers the oscillatory motion. The extracted SEC is again the same as the one stored in the action library (see Figure 2.3).

In Figure 2.20 and Figure 2.21 we show the 3D trajectory of the robot arm for both *Put on top* and *Cutting* actions. The start and the end of the trajectory as well as position of objects in the action, are highlighted with red circles and text labels.

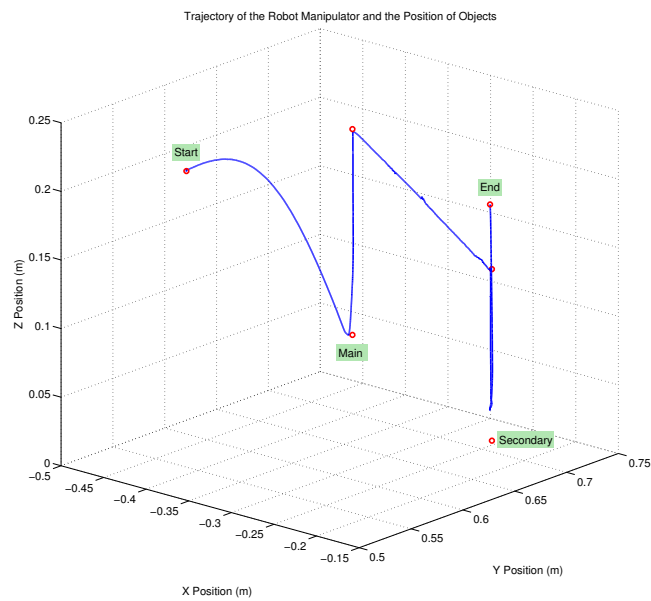


FIGURE 2.20: Trajectory of robot arm during the *Put on top* action shown in Figure 2.18.

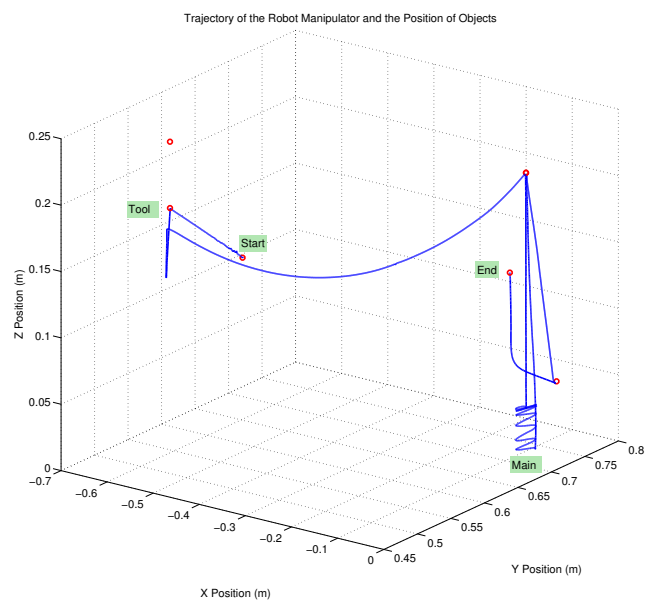


FIGURE 2.21: Trajectory of robot arm during the *Cutting* action shown in Figure 2.19.

2.4.3 Chained Actions

To demonstrate the scalability and strength of the proposed framework, we further benchmarked our system with execution of chained actions. For this purpose, we defined two scenarios. In the first scenario, the robot arm was given the task of performing three atomic actions: *Take down*, *Push*, and *Put on top*. The second scenario is a more challenging task: *making a salad*.

Figure 2.22 shows the robot execution of the first chained action scenario. The first three plots depict the low-level sensory data. Due to having three atomic actions, there exist three peaks in the force sensor, whereas we obtain only two contact changes in the tactile sensor. In each action there is one interval at which the contact in Z axis is detected. However, we can see that in the second action there is no grasping.

In the second scenario, i.e. the salad making task, the robot performed a longer action sequence, in which we additionally introduced the last two actions defined in Table 2.4: pouring and unloading. Consequently, the salad scenario contains the following steps:

1. Pick up a cucumber and *put it on* a cutting board
2. Grasp the knife and *cut* the cucumber
3. Grasp the cutting board and *unload* the cucumber pieces into a bowl
4. Grasp the bottle and *pour* its content into the bowl
5. Grasp a spoon and *stir* ingredients in the bowl

The final results of the salad scenario are shown in Figure 2.23. The low-level signals and high-level symbolic object relations are shown as usual. For the sake of clarity, sample snapshots for all five actions are shown vertically with horizontal arrows on the top showing the corresponding temporal interval of each action.

Note that in both scenarios, we assume that the high-level action plan is given in advance since we are not addressing any planning related issue in this study. Our only aim is to introduce a generic representation for the seamless execution of atomic and sequential actions independent from variations in the scene context.

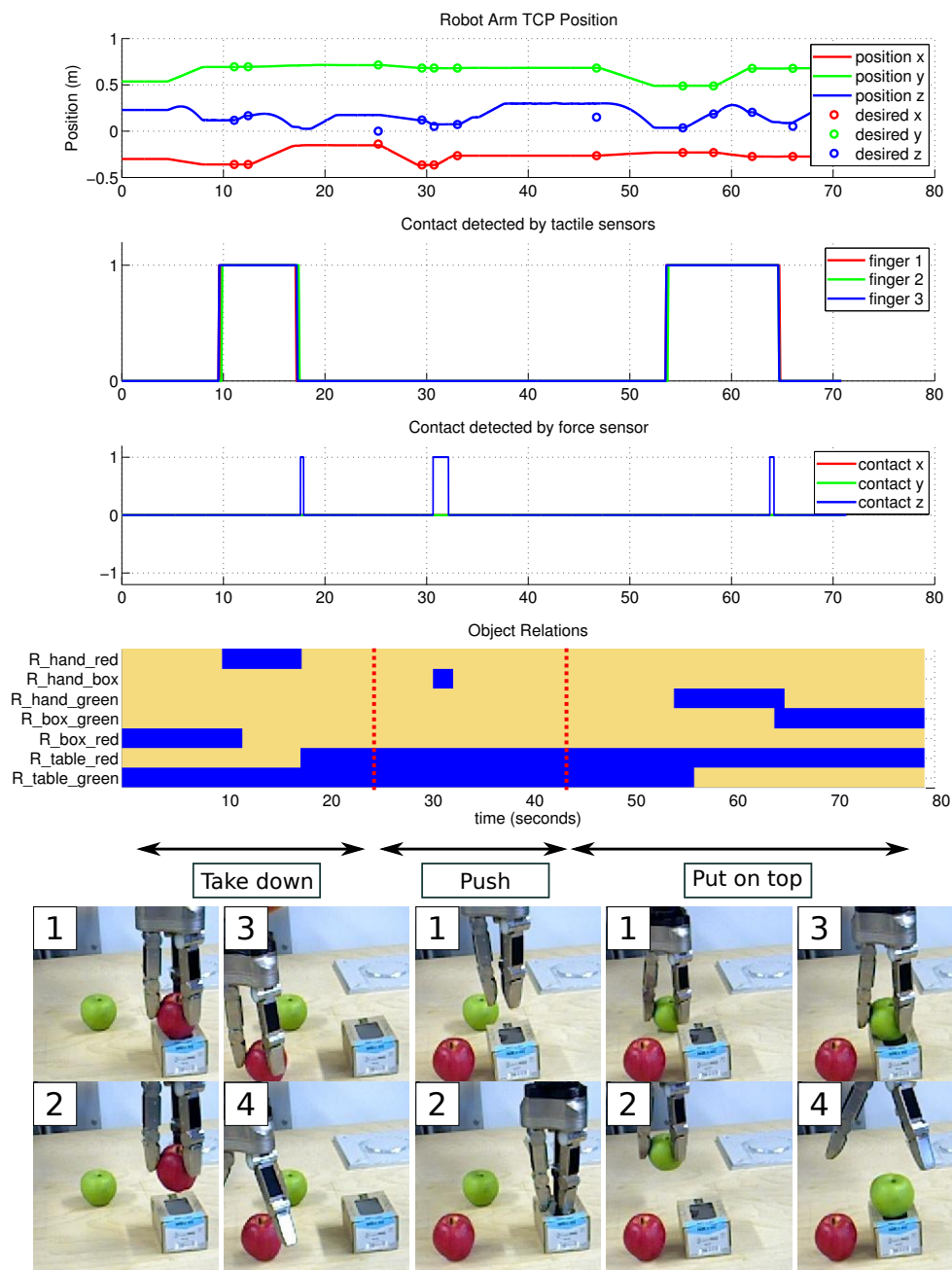


FIGURE 2.22: Robot execution of three chained actions: 1- Taking down the red apple from the box 2- Pushing the box by holding. 3- Putting the green apple on top of the box. From top to bottom are shown the position, tactile, and force sensor data as well as the changes which are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted with numbers showing their order.

Black arrows represent temporal interval of each action.

The supplementary video is provided to show the execution of single and chained actions and the error handling examples. (see the following page: <https://sites.google.com/site/aeinwebpage/actions/videos>)

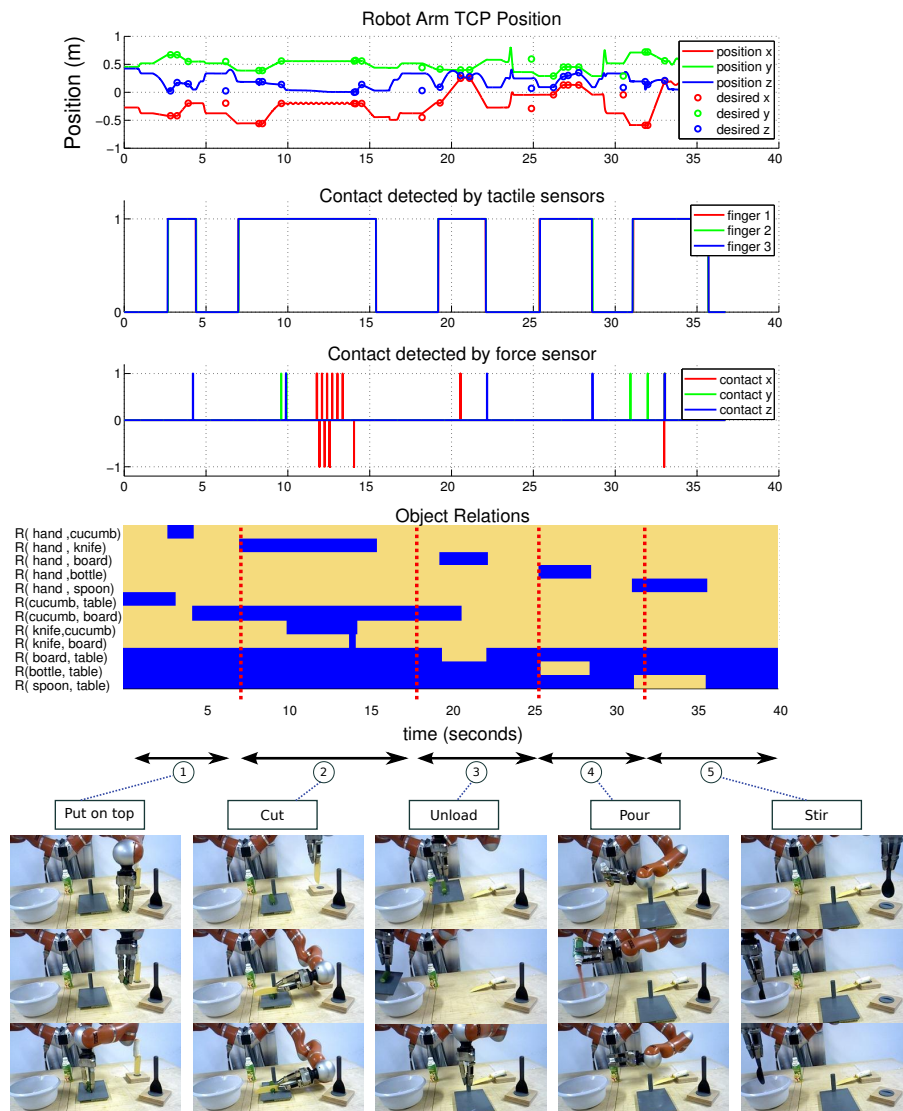


FIGURE 2.23: Robot execution of a salad preparation scenario which involves 5 atomic actions: 1- Put on top 2- Cut 3- Unload 4- Pour 5-Stir. From top to bottom are shown the position, tactile, and force sensor data as well as the changes which are detected in the relation of objects in the scene. Sample snapshots for some SEC states are also depicted at the bottom. Black arrows represent temporal intervals for atomic actions.

2.5 Conclusion

In this chapter we introduced a novel definition of manipulation actions with both symbolic and sub-symbolic components. We also introduced a mechanism for executing them on a generic robot arm/hand manipulator. We showed the execution results for single actions as well as longer tasks in chained actions. (For more details of single action results see [Appendix B](#))

While actions are successfully executed with a variety of objects and poses, we have also failures. By applying the actions on more objects with different sizes and poses, we can separate the random failures from systematic ones and reach some conclusions about object-action affordances. Random failures can be solved by altering the parameters of the action execution. This motivates us to use parameters beyond the predefined ones used so far. In the next chapter, we investigate two methods of improving the parameters from human demonstrations and by using previous experiments.

The majority of this chapter and results are from our submitted paper [\[53\]](#).

Chapter 3

Bootstrapping Action Execution

In Chapter 2 we presented a multi-level definition and a mechanism for execution of manipulation actions. In this chapter we want to enhance the performance of action execution by incorporating the concept of *structural bootstrapping* [54].

Structural bootstrapping is an idea based on the concept of structural and syntactic bootstrapping which exists in child language acquisition literature [55–57]. In linguistics it means that children have innate language learning abilities in form of internal models. In the context of robotics and AI, bootstrapping attempts to increase the speed and efficiency of learning by building generative models, using existing structures or “scaffolds” to learn novel concepts. This approach enables generalization and learning of new skills from a small set of training data such as the method of [16]. Structural bootstrapping can be employed at both high- and low-level components. We use our action definition as the scaffold for bootstrapping.

We apply bootstrapping to go beyond actions with pre-programmed parameters. We develop two mechanisms for bootstrapping, one which includes human demonstrations and another by means of semantic compilation of previous experiences.

In the first mechanism, we propose a method to obtain the parameters of actions from human demonstrations. We perform the human demonstration by different methods and record the data from various sensors. Then by processing these data, we calculate a sequence of primitives. These primitives are stored in a generalizable way, and can be executed in a different situation.

The second bootstrapping mechanism aims at combining and re-using parameters from previous experiments, to execute new instructions. An algorithm is developed for this and used in some illustrative examples. We call this process “compilation”. To implement compilation we use a special data structure called action data table (ADT) to represent and store a single action execution. An ADT file is an XML file defined with a grammar which includes both high- and low-level action parameters. This format is closely related and compatible to the proposed action definition and ontology in Chapter 2. While action definition and ontology define actions in general, an ADT stores the data for a specific example of an action with certain objects and poses.

This chapter is organized as follows. In Section 3.1 we give an overview of data acquisition method. Next, in Section 3.2 we present an algorithm to automatically obtain action parameters from human demonstrations. In Section 3.3 we show how we can *compile* new actions by combining data from existing actions. We show some results in Section 3.4 and conclude this chapter by a discussion in Section 3.5.

3.1 Data Acquisition

To perform bootstrapping, we need to analyze human demonstrations of action. In this section we describe two methods to record data from human demonstrations. First, we describe how we record actions by kinesthetically guiding the robot manipulator. Second, we show that we can record similar data in an augmented reality environment.

3.1.1 Demonstration by Kinesthetic Guidance

Kinesthetic guidance is a method of human-robot interaction, in which the human operator holds the robot and performs a task. By recording the trajectories of the robot arm, it is possible to repeat the exact same task. Recent progress in building compliant and kinematically redundant robots makes this kind of human-robot interaction more important, since explicit modeling of these manipulators becomes more difficult. At the same time, kinesthetic guidance becomes easier thanks to lightweight and compliant manipulators.

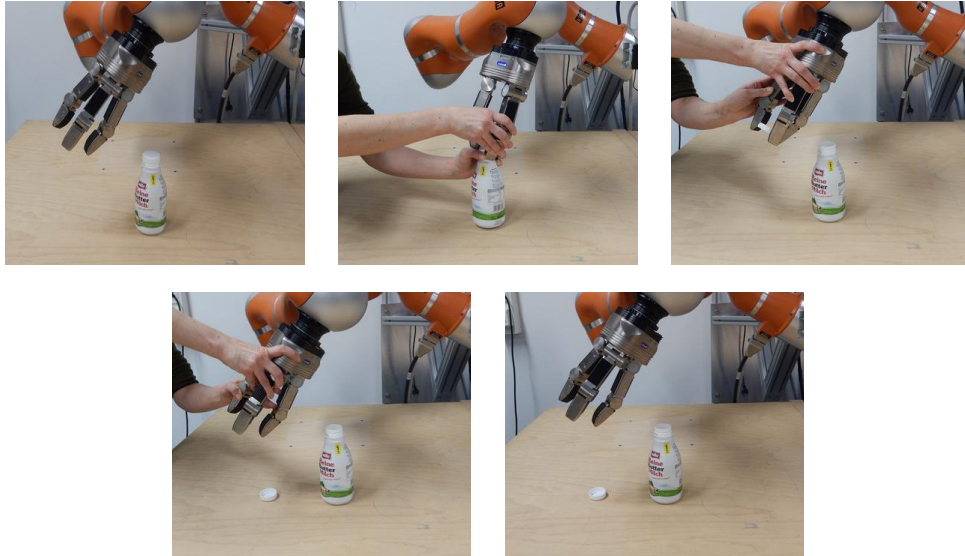


FIGURE 3.1: The operator performs a kinesthetically guided demonstration. The bottle cap is being unscrewed from the bottle.

Since we use an RGB-D sensor to record point clouds, we record the data in two epochs: once with the human operator, and once without.

1. First the human operator performs a task (like unscrew the cap of a bottle shown in Figure 3.1) by holding the robot arm and moving it. The operator moves the joints of robot hand for grasping or re-shaping the hand. The robot arm is made compliant by setting its stiffness parameters to low values, so that the operator can move it around easily. In this epoch the robot arm and hand joint values are recorded so that the same task can be reproduced without the human.
2. In the second we perform the same action by replaying the recorded trajectories. This time we record the point cloud with RGB-D sensor as well as all the available data in configuration and task space. We record the arm joint angles and torques, arm end effector pose and force, and the hand joint angles and tactile sensor output. All of these data are stored in a file and stored for each demonstration. Since we use ROS [58] as our communication framework, we store all data in ROS data format (Rosbag).

The data recorded in the *unscrew* example is shown in Figure 3.2. These can be used to do the exact same action on the same objects in the same positions. However to do any kind of generalization, we need to analyze these data to find the semantics of actions.

We do this by using our action definition framework and obtaining their parameters from the recorded data in Section 3.2.

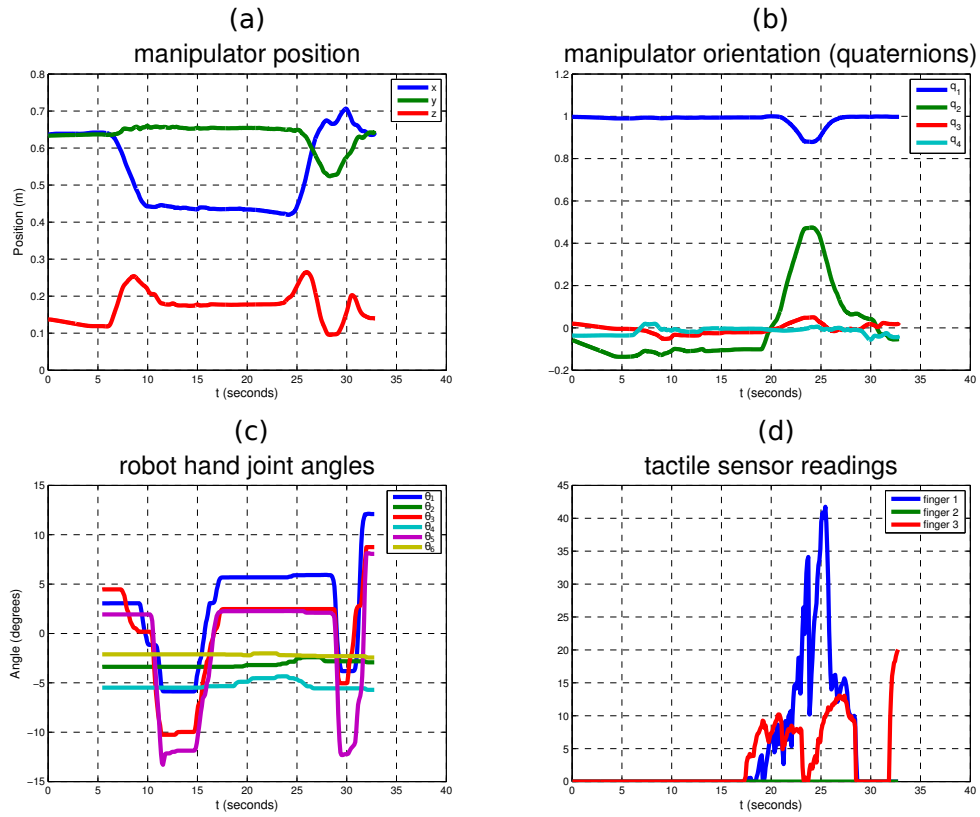


FIGURE 3.2: The data recorded in the demonstrated *Unscrew* action of Figure 3.1. (a),(b) Cartesian position and orientation of the end effector of the manipulator. (c) The joint angles of the robot hand. (d) The tactile sensor readings during the action.

3.1.2 Demonstration in Virtual Reality

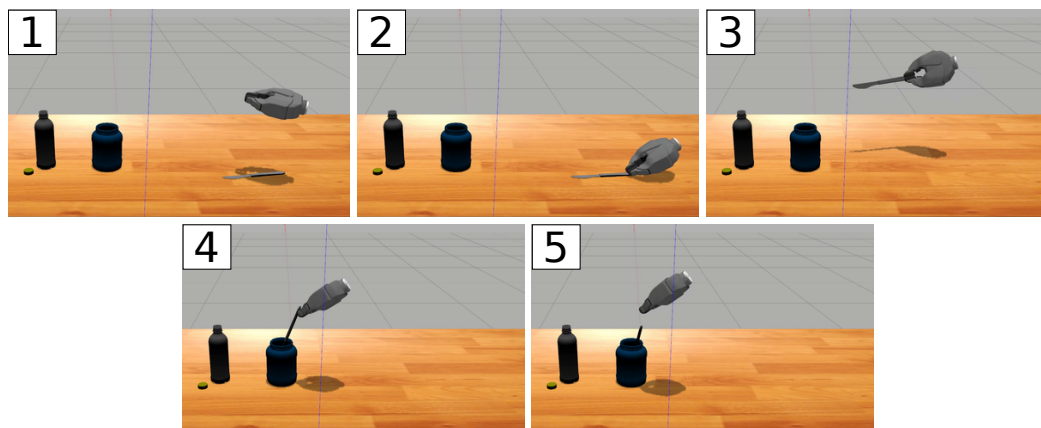


FIGURE 3.3: An example of *Insert* action demonstrated in the simulated environment.

The second method we use for recording human demonstration is a virtual reality environment introduced in [59]. Here the gazebo [60] simulator is used to benefit from its realistic physics engine. The movements of operator's hand are projected to a simulated hand via a motion sensing game controller called Razer Hydra. The pose of objects (including the robot gripper) and their touching relations are recorded at each time-step of the simulation.

An example of *Insert* action performed in this environment is shown in Figure 3.3 The data obtained in this demonstration are shown in Figure 3.4.

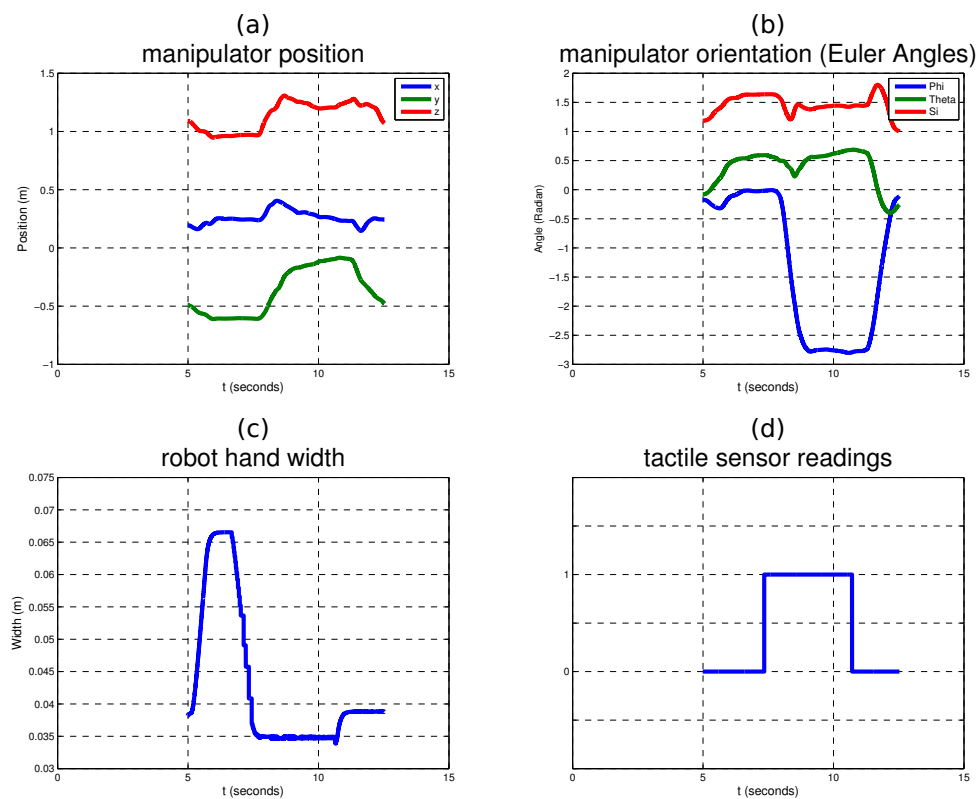


FIGURE 3.4: The data recorded in the *Insert* example demonstrated in the simulated environment. (a),(b) Cartesian position and orientation of the end effector of the manipulator. (c) The width of the robot hand. (d) The tactile sensor readings during the action.

3.2 Bootstrapping from Human Demonstration

The goal of this section is to process the recorded data and obtain generalizable descriptions for the actions. From each experiment, we are looking for three components:

- SEC matrix
- Object Roles
- Action Primitives

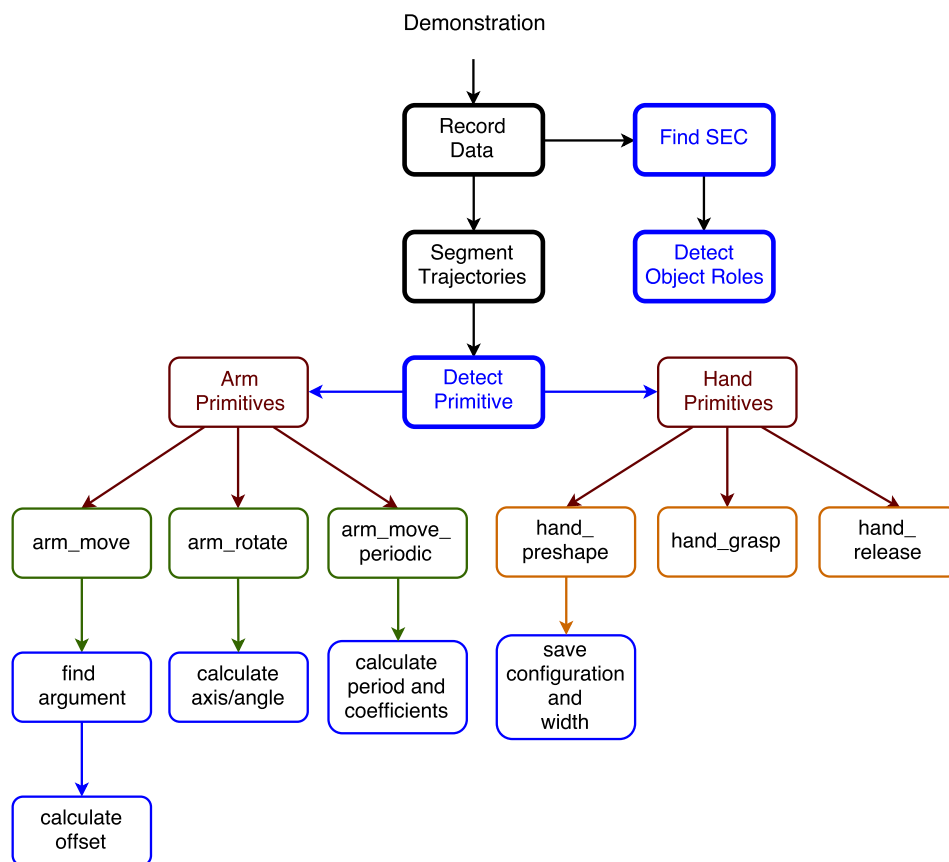
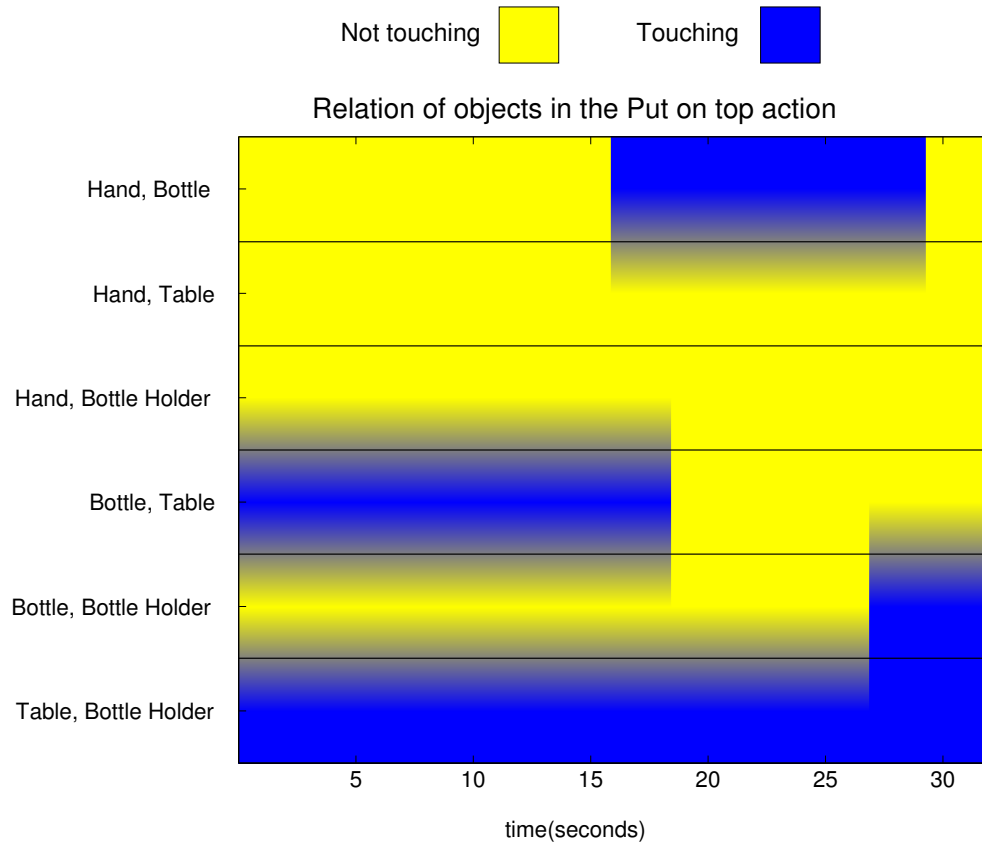


FIGURE 3.5: The process of finding action description from human demonstrations. The demonstrated action is recorded and three main components are calculated: 1-SEC matrix 2-Object roles 3-Action primitives. The trajectories are further analyzed to find the types, arguments and parameters of the primitives.

The block diagram of this process from demonstration to finding the parameters of primitives are shown in Figure 3.5 In the following sub-sections we explain the methods to calculate these components from the recorded data.

3.2.1 Calculating the SEC matrix

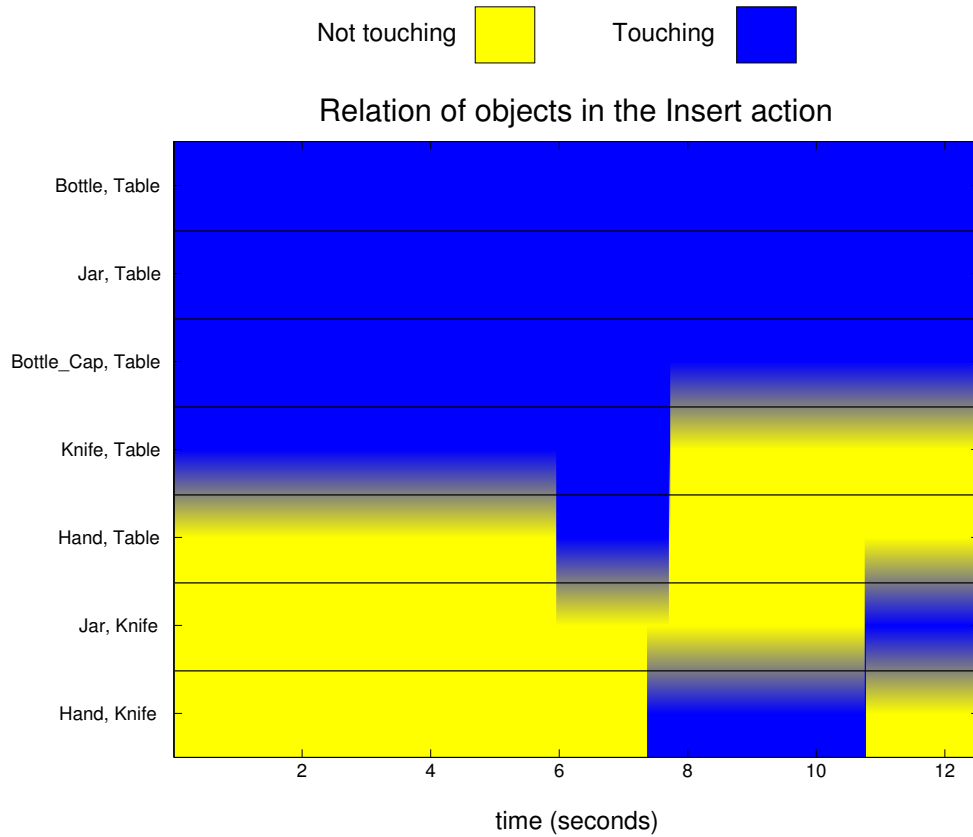


Relation

R (Hand, Bottle)	N	T	T	T	N
R (Hand, Table)	N	N	N	N	N
R (Hand, Bottle Holder)	N	N	N	N	N
R (Bottle, Table)	T	T	N	N	N
R (Bottle, Bottle Holder)	N	N	N	T	T
R (Table, Bottle Holder)	T	T	T	T	T

FIGURE 3.6: The relations of objects during the demonstrated *Put on top* action of Figure 3.8 are shown. The SEC matrix is also derived.

The SEC matrix is introduced in Section 2.1.1.2. To calculate the SEC matrix we should track the touching relations of objects in the scene. In the kinesthetically guided demonstrations we use our vision system which segments and tracks objects [61]. Each object is represented by a labeled 3D point cloud and the touching relations are calculated by estimating the minimum distance between two objects. The relations are monitored throughout the action. For our *Put on top* example of Figure 3.8, the resulting relations and SEC matrix are shown in Figure 3.6.



Relation

R (Bottle, Table)	T	T	T	T	T	T	T
R (Jar, Table)	T	T	T	T	T	T	T
R (Bottle Cap, Table)	T	T	T	T	T	T	T
R (Knife, Table)	T	T	T	T	N	N	N
R (Hand, Table)	N	T	T	N	N	N	N
R (Jar, Knife)	N	N	N	N	N	T	T
R (Hand, Knife)	N	N	T	T	T	T	N

FIGURE 3.7: The relations of object during the demonstrated *Insert* action of Figure 3.3 are shown. The SEC matrix is also derived.

In the simulated experiments, we have the exact poses of the objects in the scene and calculating the relations is easier. The relation of objects and the SEC matrix for the *Insert* example are shown in Figure 3.7.

3.2.2 Calculating the Object Roles

Here we want to find the role of each object in the action. The roles are defined in Table 2.1. We find the roles based on only the relation changes, in a similar way to the

method in [17]. Note that we do not assume that we know any of the objects, not even the *manipulator*.

We follow the following steps and show this again by the *Put on top* example:

- The *manipulator* and *main* should make a sequence of $N T N$ changes. We can see that the only relation that has such a sequence is the relation of hand and bottle. First we assume that the hand is the *main* object. We check the other relations involving the hand: $R(\text{hand}, \text{bottle holder})$ and $R(\text{hand}, \text{table})$. Since these relations are both constantly N , the hand can not be the *main* object. Therefore the hand is the *manipulator*. Therefore we conclude that the bottle is the *main* object.
- The *primary* object should make a $T N$ transition to the *main*. This defines the role of *primary*.
- The *secondary* object should make an $N T$ transition with the *main*, which means it is the bottle holder.
- The bottle holder is also touching the table, which means that the *primary* and *secondary* object are also touching. By comparing the positions of bottle holder and table, we find that the bottle holder is on top of the table. Therefore, the *primary* object is also the *secondary support*.

For the *Insert* example of Figure 3.3, we can find the object roles in a similar way:

- We look for $N T N$ changes in the relations. The relations $R(\text{hand}, \text{knife})$ and $R(\text{hand}, \text{table})$ both have such transitions. We can first conclude that hand is the *manipulator* since it does not make a touching relation at the beginning and the end of the action.
- Both table and knife could be our *main* object but since the knife has both $T N$ and $N T$ transitions, we know it is the *main*. We note also that table has several other constant T relations with the bottle, the jar and the bottle cap.
- Since the knife is the *main*, we can identify table and jar to be *primary* and *secondary* object respectively.

TABLE 3.1: The object roles of two examples are obtained by analyzing the object relations. The *Put on top* example is shown in Figure 3.8 and the *Insert* action is performed in the simulation environment of Figure 3.3.

Object Role	Put on top	Insert
manipulator	hand	hand
main	bottle	knife
primary	table	table
secondary	bottle holder	jar
secondary support	table	table

- The table is also the *secondary support* since it constantly touches the jar.

The object roles of these two examples are summarized in Table 3.1. This type of simple geometric reasoning generalizes to all analyzed actions and, thus, we can always find the different object roles.

3.2.3 Calculating the Action Primitives

We process the trajectories and sensor data from the recordings to find primitives like *arm_move(main)*, *arm_rotate()* and *hand_grasp()*. We are also interested in their parameters, timings and the column of SEC matrix they belong to. The demonstrated *Put on top* action shown in Figure 3.8 will be used as an illustrative example.

We obtain the primitives in two steps described below.

1. Segmentation of the recorded trajectories.

The robot arm and hand trajectories are divided into segments and a primitive is assigned to each of them. The criteria for segmentation is the direction of movement. This means that within each segment, we want to have a motion towards the same target.

2. Obtaining generalizable primitives from segmented data.

The goal of this step is to determine the type of the primitives and calculate their parameters. For the robot arm primitives, we want to distinguish *arm_move()* and *arm_rotate()* primitives. Furthermore for the *arm_move()* primitives we want to determine the target objects (main, primary, etc.) and calculate the offset transforms T_{off} . In case of *arm_rotate()* we would like to calculate the axis and angle of rotation.

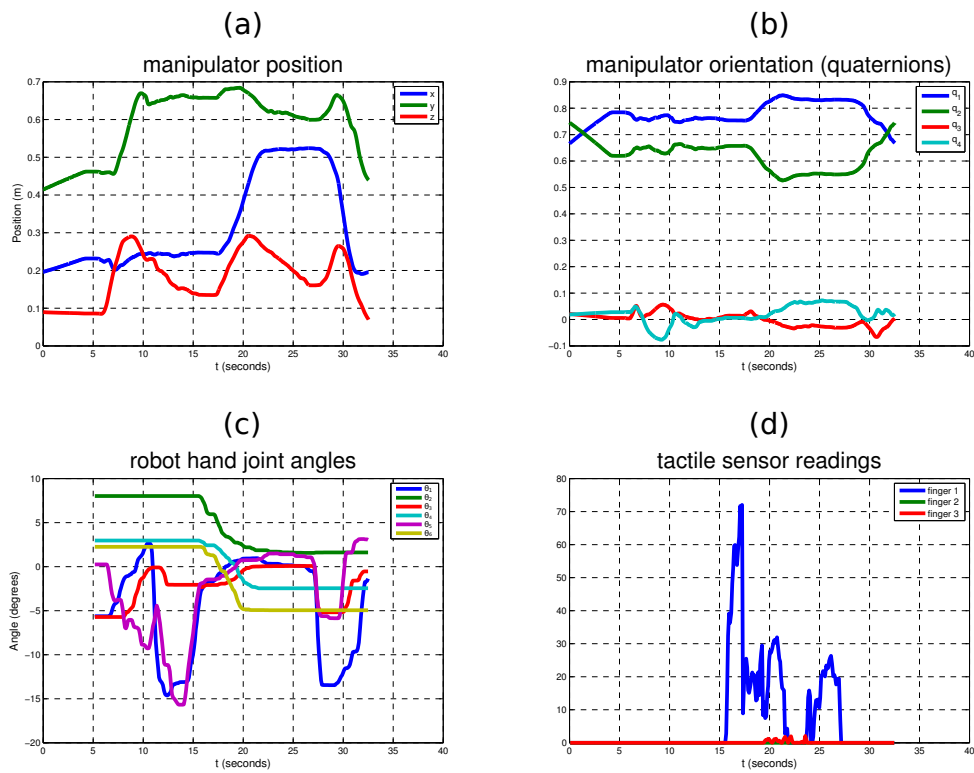
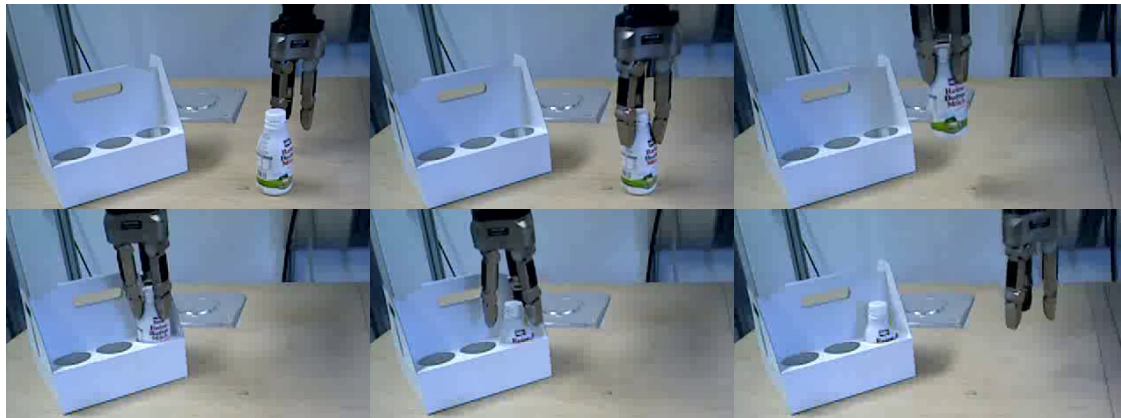


FIGURE 3.8: The data recorded from the demonstrated *Put on top* action are shown. At the top, the snapshots of the demonstration without the operator (second round) is shown. The end effector position and orientation are shown in plots (a) and (b). For the robot hand, the joint angles and tactile sensor readings are shown in plots (c) and (d).

For the primitives of the robot hand, we would like to distinguish between *hand_preshape()*, *hand_grasp()* and *hand_release()* and find their parameters.

3.2.3.1 Segmentation of Trajectories

Segmentation of trajectories is a difficult task specially in presence of sensor noise and vibrations in human demonstrations [62–64]. We use a method similar to [63] in which a

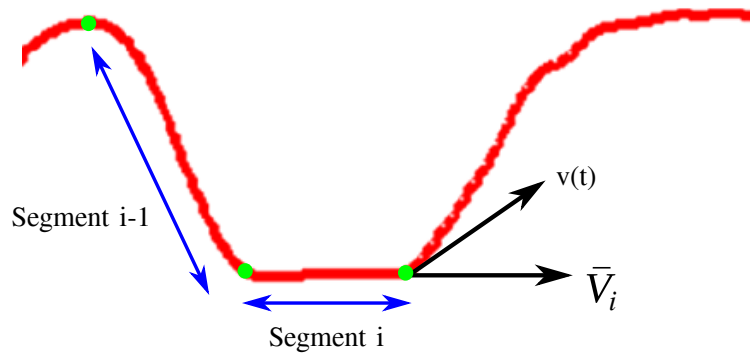


FIGURE 3.9: A simple 2D trajectory is segmented by comparing the velocity directions. The velocity at $v(t)$ diverges from the average velocity of segment i , \bar{V}_i , therefore a new segment is created. Green circles show the segmentation points created by using Equation 3.2.

trajectory is divided into segments with similar velocities. The velocity in each segment is not necessarily constant, but shows some variance. We take into account only the direction of the motion and not its amplitude.

To find the points of segmentation, we calculate the cosine of the angle between the velocity of the current point $v(t)$ and the average velocity of the current segment \bar{V}_i . If we call the angle $\theta(t)$ we have:

$$\cos(\theta(t)) = \frac{v(t) \cdot \bar{V}_i}{\|v(t)\| \|\bar{V}_i\|} \quad (3.1)$$

The condition to create a new segment is that for a threshold value $-1 < k < 1$ we have:

$$\cos(\theta(t)) < k \quad (3.2)$$

Otherwise the current point with velocity $v(t)$ is part of the current segment, and we update \bar{V}_i . An example trajectory segmented with this method is shown in Figure 3.9.

Real robot trajectories are noisy and have more dimensions, but we can segment them with a similar approach. If the velocity signals are available we use them directly, otherwise, we have to calculate them from pose¹ measurements. After applying a low-pass filter to remove noise, we differentiate the pose signal to obtain the velocities.

¹Here we use the term pose as combination of position (x,y,z coordinates) and orientation (represented by quaternions).

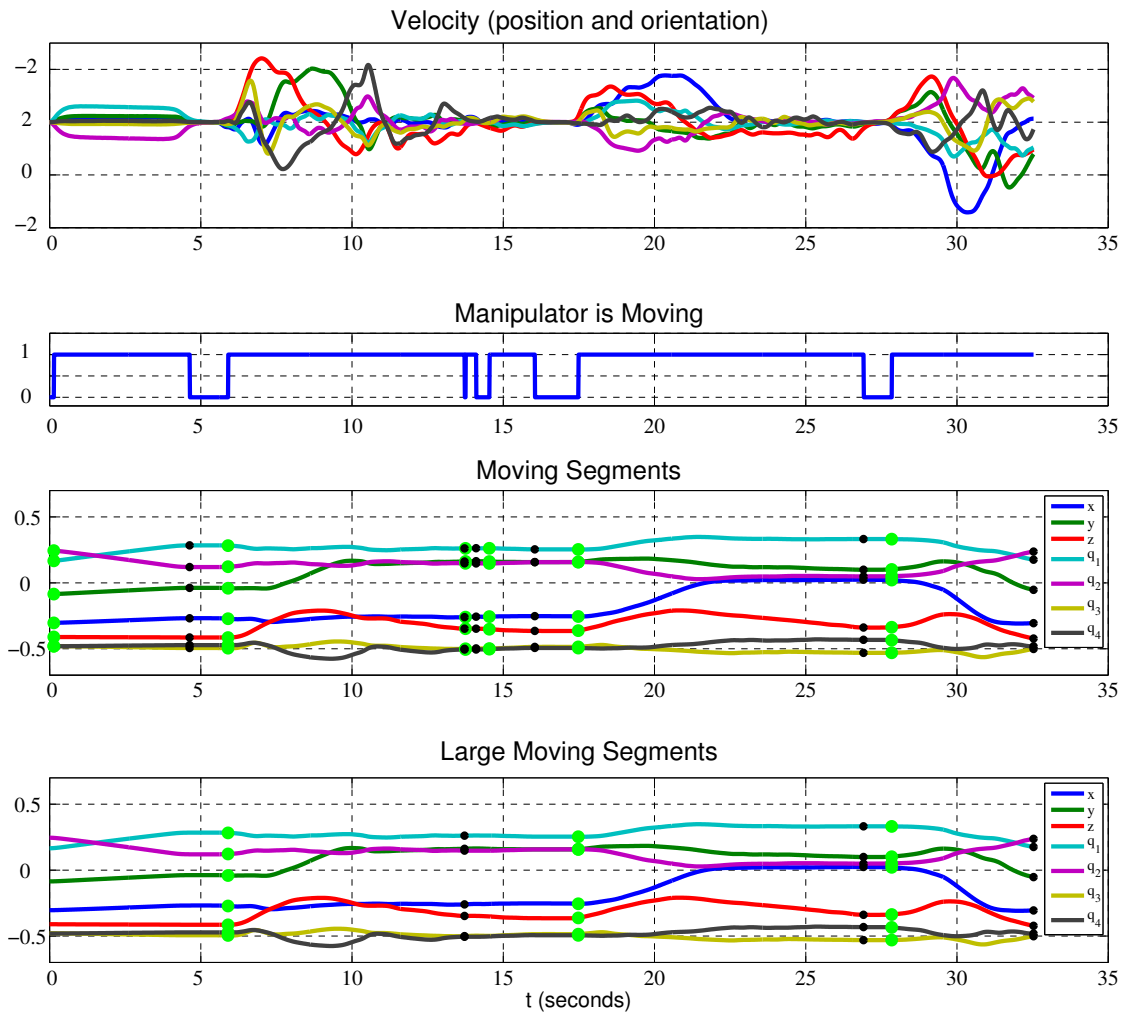


FIGURE 3.10: Initial segmentation of arm pose trajectories. The velocities are shown on the top plot. The moving parts of trajectory are shown in the second plot. The resulting segments are shown in the third plot. The last plot shows 3 large segments which are kept after removing small (noisy) motions.

The velocities of the *Put on top* example (See Figure 3.8) are shown in the top plot of Figure 3.10. By applying a threshold, we detect the parts where the robot manipulator is moving. This binary signal is shown in the second plot. The start and the end of moving parts are marked with green and black circles on the trajectory in the third plot of Figure 3.10. We note that some of these movement segments result in very small displacements. These small motions are usually unwanted vibrations generated by the operator during the demonstration. We remove these segments and keep the 3 large segments which are shown at the bottom of Figure 3.10.

The pose trajectory and its large segments are depicted again on top of Figure 3.11. However, within each segment the direction of motion may have changed and we can further divide them into smaller segments. To do this, we calculate the direction of

velocity at each point and apply Equation 3.2 to detect segmentation points. The cosine of the velocities are shown in second plot of Figure 3.11. The segmentation is done with $k = 0.8$ and the resulting sub-segments are shown in the third plot of Figure 3.11. Finally, by removing the small sub-segments, the segmentation is done and the result is shown in the last plot of Figure 3.11. Now we assert that each of these large sub-segments corresponds to one arm primitive. The average velocities are shown in Figure 3.12. The average velocity of each segment is superimposed at the middle of the segment on the 3D pose trajectory.

We use the same method to segment the robot hand trajectories. In Figure 3.13, the hand joint velocities are shown along with a flag showing the movement of hand. The segments and the large segments of the trajectories are shown in the last two plots. At this step the hand trajectory is divided into 3 segments. However, we can further divide it to sub-segments by analyzing the direction of the velocities.

Starting from the 3 segments, we calculate the cosine of the velocity direction at each point and generate a sub-segment when the cosines are below $k = 0.3$. Note that for the hand trajectories, a lower threshold is selected since the changes in the hand angles are more abrupt than the changes in arm pose. The resulting subsegments are shown in the third plot of Figure 3.14. By removing the small sub-segments, the segmentation of hand trajectories is completed and the final 6 sub-segments are shown in the last plot of Figure 3.14.

To sum up the segmentation, we have segmented the arm pose trajectory into 10 and the hand angle trajectory into 6 segments. Each segment is a candidate for a primitive. In the next step, we find out the type of each primitive and obtain its parameters.

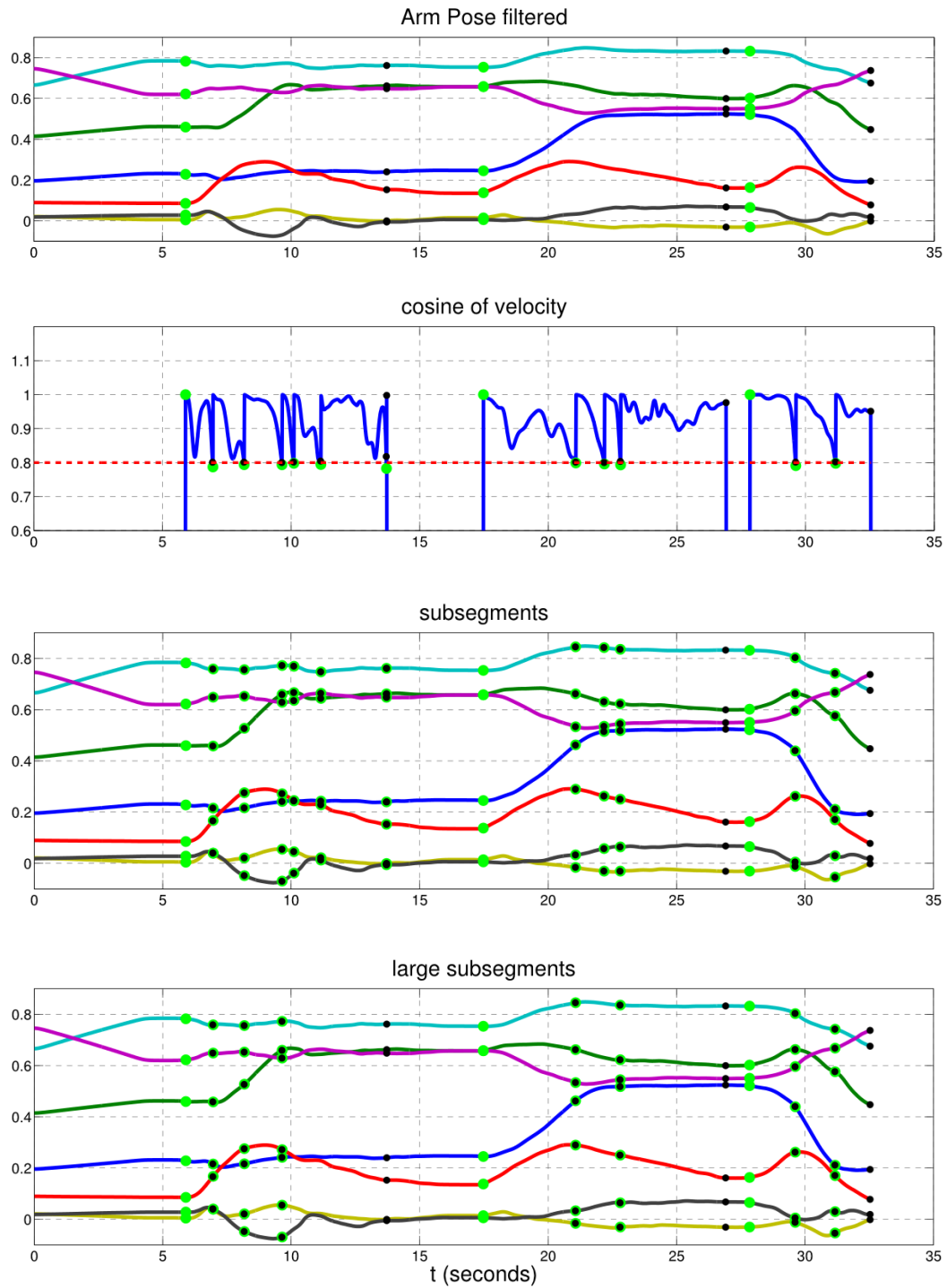


FIGURE 3.11: The segmentation of arm pose trajectories using direction of velocities. The 3 large segments (top) are divided into subsegments with different velocity direction. The cosine of angle between velocities are calculated according to Equation 3.1 and a threshold $k = 0.8$ is applied. The subsegments are shown in the third plot. After removing the small subsegments, the final segmentation of pose trajectories are shown in the bottom plot with 10 segments.

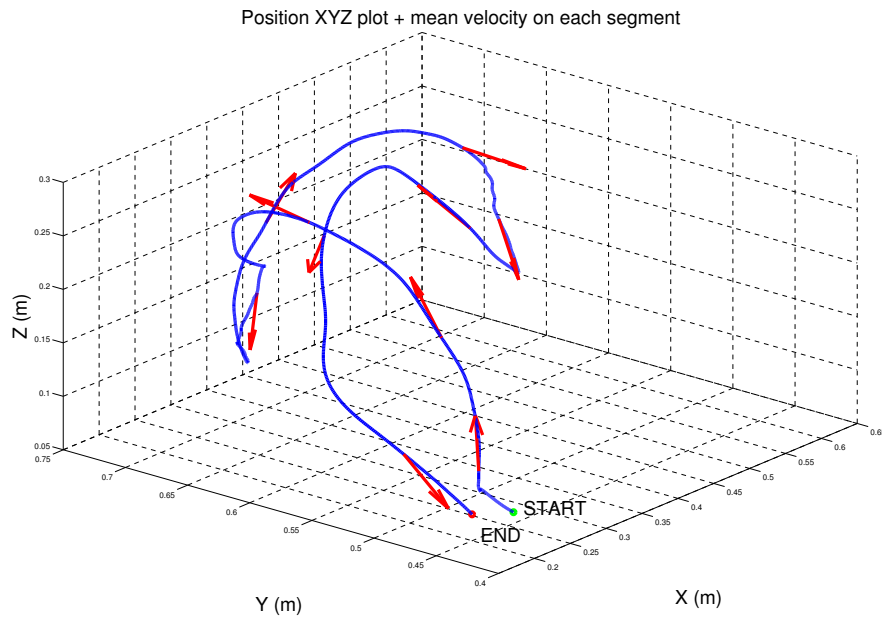


FIGURE 3.12: The position trajectory is plotted in 3D space and the start and end points are marked. The average velocity of each segment is superimposed at its middle point. The trajectory for this action is divided to 10 segments.

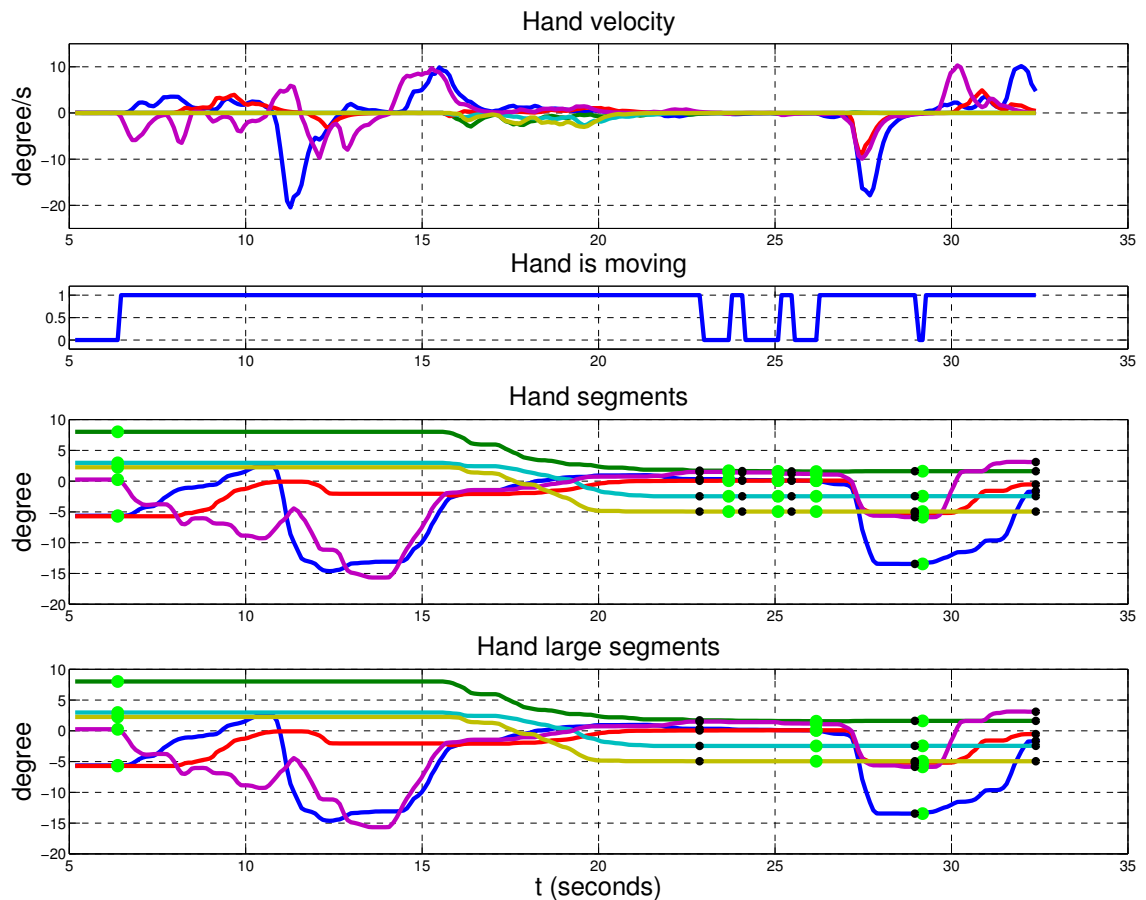


FIGURE 3.13: Segmentation of the trajectories of robot hand. The velocities are calculated (top plot) and the *hand-is-moving* flag is shown (second plot). Based on this flag, the trajectory is segmented (third plot) and after removing the small segments, 3 segments are detected.

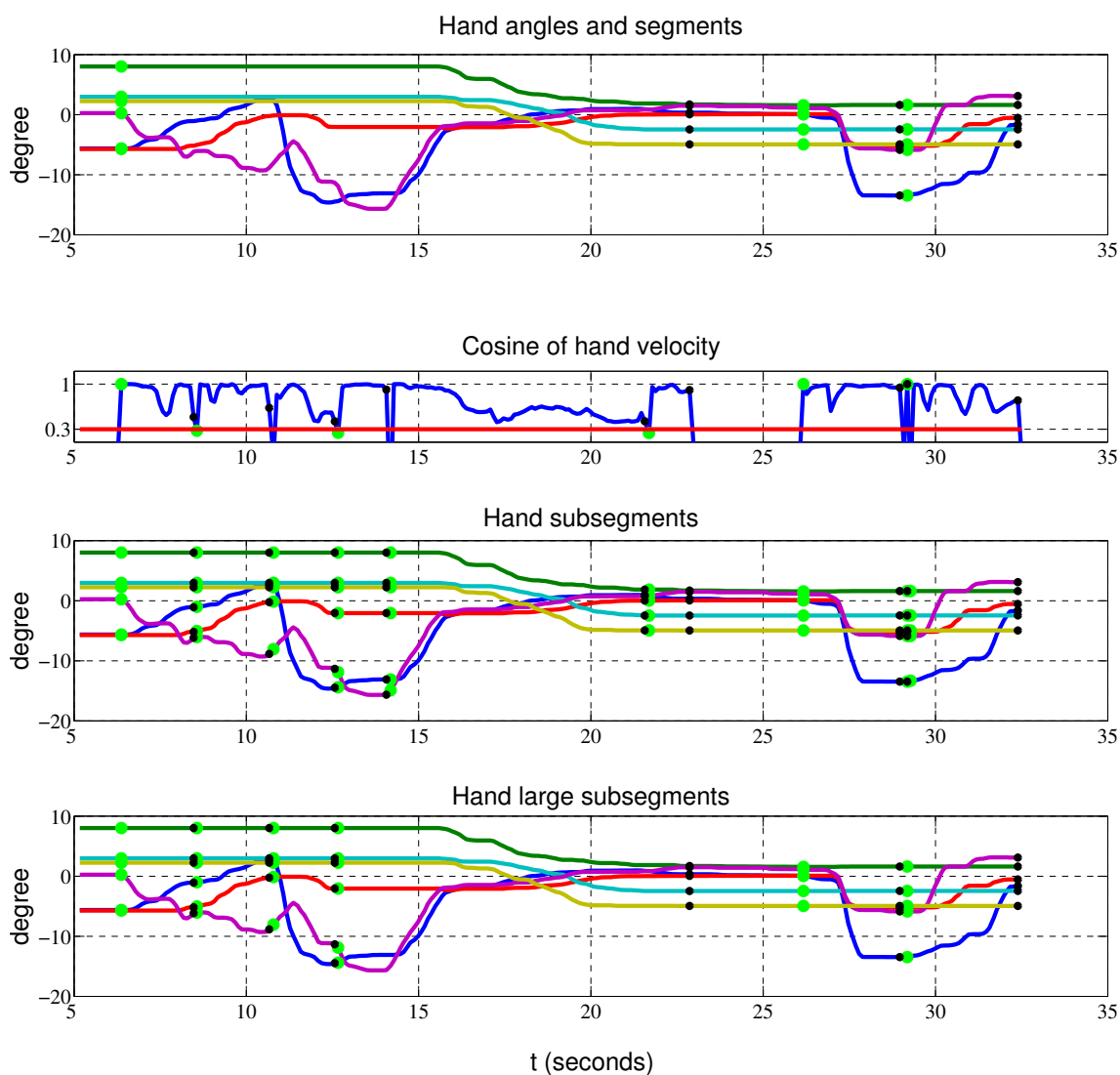


FIGURE 3.14: The 3 segments of Figure 3.13 can be further divided into sub-segments. The cosine of velocities is calculated and the segmentation points are calculated based on Equation 3.2 with $k = 0.3$. (second plot) The resulting sub-segments are shown in the third plot. After removing the small sub-segments, the final segmentation of hand trajectory is shown in the last plot.

Segment	Translation (m)	Rotation (Degree)	Primitive type
1	0.0819	2.6585	arm_move
2	0.1278	5.1108	arm_move
3	0.1344	4.9733	arm_move
4	0.1192	7.4771	arm_move
5	0.2646	2.0798	arm_move
6	0.0784	0.3266	arm_move
7	0.0922	4.4175	arm_move
8	0.1415	4.9561	arm_move
9	0.2595	5.0077	arm_move
10	0.1590	2.6643	arm_move

TABLE 3.2: Translational and angular distances of each segment of arm pose trajectory.

3.2.3.2 Identify Arm Primitives

In the previous section we segmented the arm pose trajectories. Now we want assign an arm primitive to each segment and gain more information about them. We want to detect which arm primitive (See Section 2.1.2.3) best fits to each segment. We distinguish between *arm_move()*, *arm_move_periodic()* and *arm_rotate()* primitives. Furthermore, we want to find the arguments and parameters of each primitive.

First we check if we have any *arm_rotate()* primitives. An *arm_rotate()* primitive has a negligible translation (less than 5cm) and a considerable rotation (more than 15 degrees). Table 3.2 shows the translational and rotational distances of each segment. Since none of the segments meets the criteria of *arm_rotate()*, we can conclude that all segments in this example are *arm_move()* primitives.

Next, we should find the parameters of the *arm_move()* primitives i.e. the goal object and offset transform. For this, we associate the movement of each segment to an object in the scene. In other words, we want to estimate which object in the scene has been the target of the motion of the arm in each segment.

Suppose we have N segments in the trajectory and m objects (other than the manipulator) in the scene. We use the following measures to detect the argument of the primitive:

- Direction measure: For each segment, we calculate a vector L_i from the start to the end:

$$L_i = E_i - S_i \quad i = 1, \dots, N \quad (3.3)$$

where E_i and S_i are the position of the arm at the end and beginning of segment i . We also calculate the vectors from start of each segment to each object.

$$a_{ij} = P_j - S_i \quad i = 1, \dots, N \quad j = 1, \dots, m \quad (3.4)$$

where P_j is the position of the j -th object. Next we calculate the cosine of the angle between L_i and each a_{ij} .

$$c_{ij} = \frac{L_i \cdot a_{ij}}{\|L_i\| \|a_{ij}\|} \quad i = 1, \dots, N \quad j = 1, \dots, m \quad (3.5)$$

The direction measure is calculated from this cosine:

$$m_{ij}^{dir} = 1 - |c_{ij}| \quad (3.6)$$

If the movement of the manipulator in the i -th segment is towards (or away from) object j , the value of c_{ij} is close to 1 (or -1) respectively. This makes the direction measure m_{ij}^{dir} approach zero. However, there could be cases where more than one object have high c_j values. For this we also use the distance measure, so that the objects closer to the arm are preferred.

- Distance measure: For each segment, we calculate the vector from end of segment to each object:

$$d_{ij} = P_j - E_i \quad i = 1, \dots, N \quad j = 1, \dots, m \quad (3.7)$$

The distance measure is simply the magnitude of this vector:

$$m_{ij}^{dist} = \|d_{ij}\| \quad (3.8)$$

- Combined direction and distance measure: To get a better estimate of the argument object, we combine the direction and distance measures:

$$m_{ij}^{combined} = \alpha m_{ij}^{dist} + (1 - \alpha) m_{ij}^{dir} \quad 0 < \alpha < 1 \quad (3.9)$$

In our *Put on top* example, we have $N = 10$ segments and $m = 3$ objects (main, primary and secondary). The direction, distance and combined measures are shown

in Figure 3.15. Based on the combined measure (bottom plot), we can estimate the argument of each *arm_move()* primitive. For example, the argument of the first primitive is the *primary* object. The arguments of arm primitives in this example are shown in the second plot of Figure 3.17.

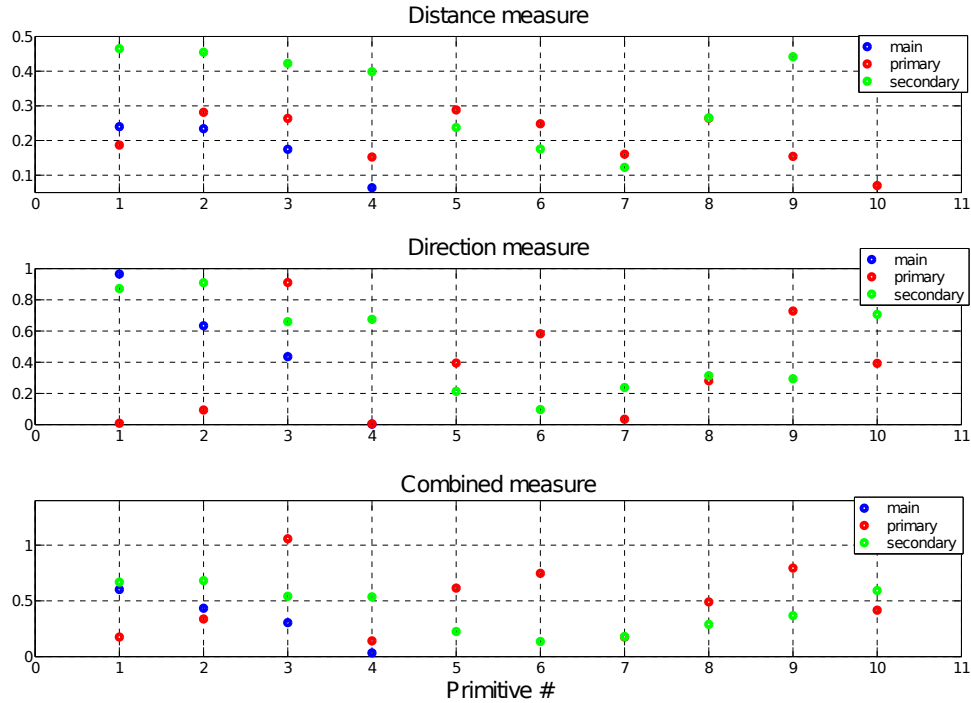


FIGURE 3.15: The direction, distance and combined measures for the *Put on top* example. The object with the lowest combined measure is selected as the argument of each primitive.

If no object could be associated to a segment, the argument of the *arm_move()* primitive is set as *free*, indicating an arbitrary pose. The exact value of the *free* poses can be set during the action, to a convenient pose.

We also calculate the relative pose of the manipulator at the end of segments to the goal object. This transform is the T_{off} parameter of *arm_move()* primitives.

3.2.3.3 Identify Hand Primitives

The type of hand primitives is determined based on joint angles and tactile sensors. The goal is to distinguish *hand_preshape()*, *hand_grasp()* and *hand_release()* primitives. The following conditions determine the primitive types:

- *hand_grasp()*: The velocity of the hand angles are negative (closing the hand) and there is a positive tactile event.
- *hand_release()*: The velocity of the hand angles are positive (opening the hand) and there is a negative tactile event.
- *hand_preshape()*: The primitive meets neither of the above conditions.

Figure 3.16 (top) shows the trajectories of hand joints superimposed by the detected segments. The tactile events are shown at the bottom plot. Based on these data, the hand primitives are identified and the results are shown in the top plot of Figure 3.17

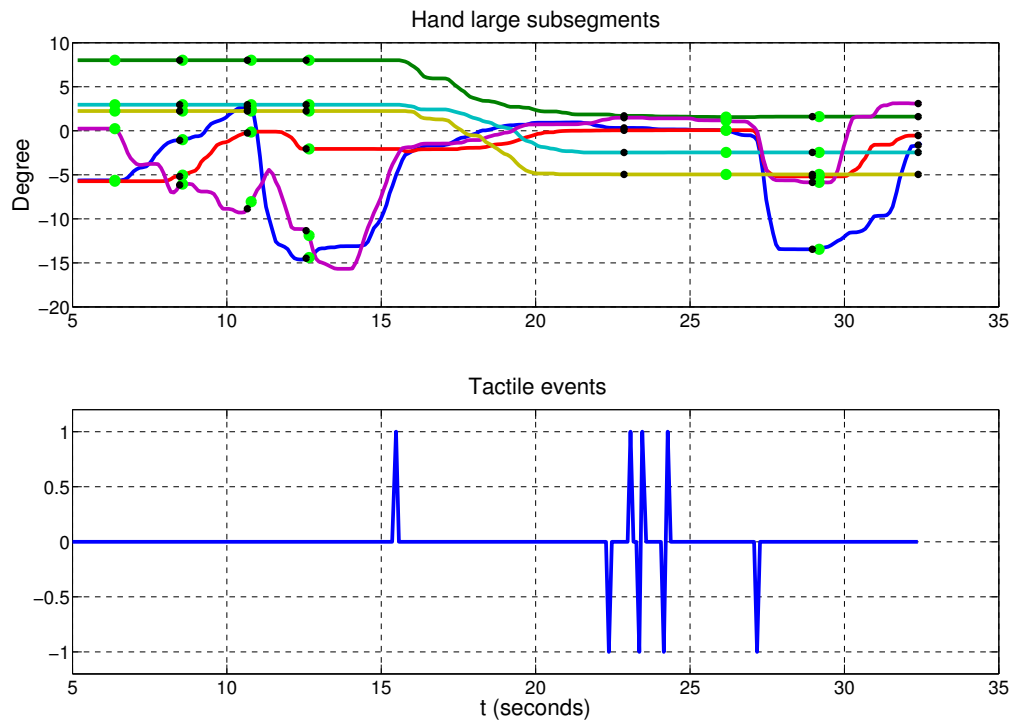


FIGURE 3.16: The primitives of hand are extracted by combining data from joint angles and tactile sensors. Segmentation of joint trajectories reveals the number and time of primitives (top). The tactile events (bottom) help us to distinguish between *hand_preshape()*, *hand_grasp()* and *hand_release()* primitives.

3.2.4 Combine Multiple Demonstrations

The above mentioned method gives a rather precise description of a single demonstration by finding its primitives. However, a single demonstration is not enough to learn a skill. Better generalization is achieved if multiple demonstrations of the same action are analyzed and the results are combined. The combination removes the noise of single

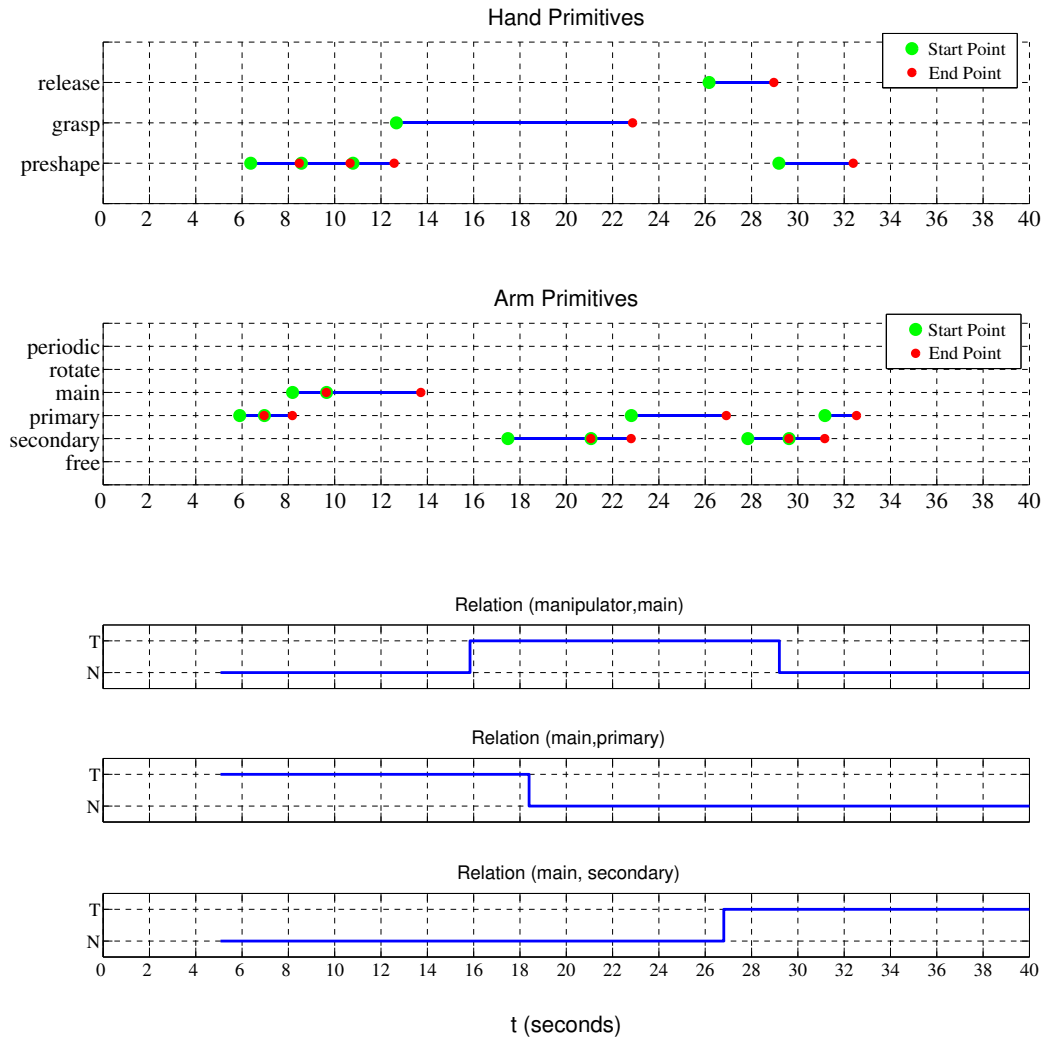


FIGURE 3.17: The obtained arm and hand primitives of the *Put on top* action demonstrated by human operator. The top and middle plots show the hand and arm primitives. Start and end times are shown by green and red circles. The time of each primitive is shown by a blue line. The type of primitives are shown in the Y-axis. For the *arm.move* primitives, the argument object is shown in the Y-axis. (free, main, primary and secondary). In the last three plots, the observed relations of objects are shown to give a better description of the action.

demonstrations and converges to the correct description of an action. This idea is applied to the *Put on top* action to generate a description based on a few demonstrations.

For this, we analyzed 6 demonstrations of *Put on top* with different objects and poses but the same SEC matrix and object roles. The snapshot of these experiments are shown in Figure 3.18.

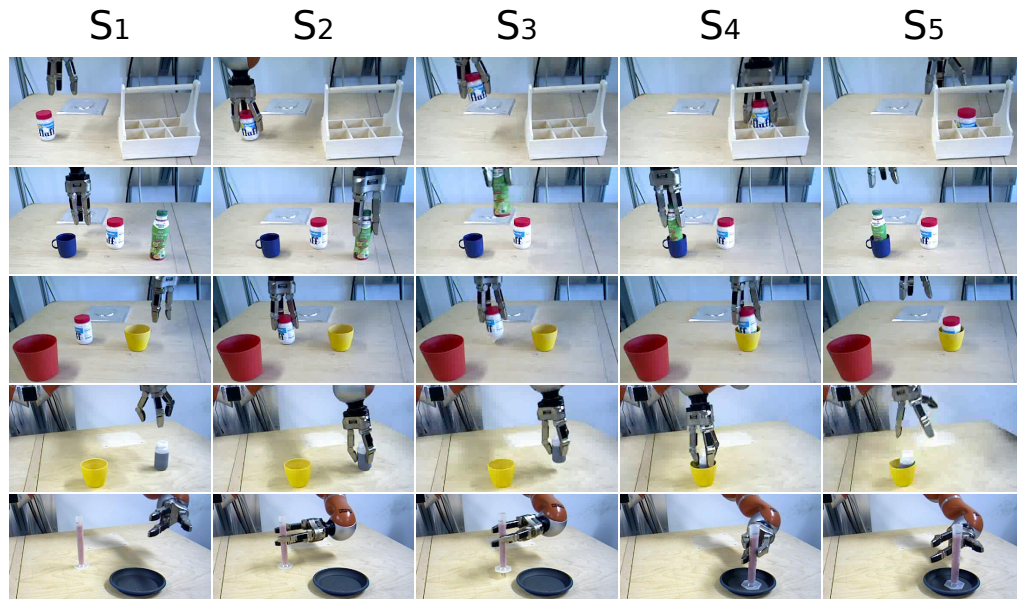
The extracted primitives of all demonstrations are shown for each column of SEC matrix in Table 3.3.

TABLE 3.3: The primitives are 6 demonstrations of *Put on top* action are compared at each state. The entries of table are the number that each primitive type and argument appeared in each demonstration. The primitives which are more frequently repeated across different demonstrations, are kept in the general description of the *Put on top* action.

state #	demo #	arm_move			hand		
		main	primary	secondary	preshape	grasp	release
1	1	2	2		3	1	-
	2	2				1	
	3	2	1		1	1	
	4	3			1	1	
	5	4			2	1	
	6	1	1			1	
2	1			1			
	2			1			
	3			1	1		
	4		1	1			
	5			1			
	6			1	1		
3	1		1	1			
	2			1			
	3			2			
	4			2			
	5			2			
	6			1			
4	1		1	2	1		1
	2			1			1
	3			2	2		1
	4		1	2	1		1
	5		1	2	1		1
	6			1	1		1

If we keep the primitives seen frequently in the demonstrations, we obtain the result shown in Figure 3.19.

It is interesting to compare this result with the pre-defined version of the same action in Figure 2.2. We can see most of the primitives are similar, except the argument of the *arm_move()* primitive in the second column and an extra *hand_preshape()* at the end. The first difference is because the movement away from primary object is usually continued toward the secondary object. The second difference comes from the human operator moving the joints of the robot hand at the end of action, which is not so crucial.

FIGURE 3.18: Human Demonstrated *Put on top* actions with different objects.

Relation/State	S1	S2	S3	S4	S5
R (manipulator, main)	N	T	T	T	N
R (manipulator, secondary)	A	A	A	A	A
R (manipulator, primary)	A	A	A	A	A
R (main, secondary)	N	N	N	T	T
R (main, primary)	T	T	N	N	N
R (secondary, primary)	T	T	T	T	T
R (main, primary sup.)	A	A	A	A	A
R (main, secondary sup.)	T	T	N	N	N
Primitive 1	hand _preshape	arm_move (sec.)	arm_move (sec.)	hand _release	-
Primitive 2	arm_move (main)	-	-	arm_move (sec.)	-
Primitive 3	hand_grasp	-	-	hand _preshape	-

FIGURE 3.19: Primitives of *Put on top* action by combining the results of all demonstrations.

3.3 Bootstrapping by Action Compilation

Our aim in this section is to improve the quality of execution by using the knowledge gained from previous experiments. In order to do this we use the structure of actions introduced in Chapter 2, and try to get the parameters from a database of executed actions which are stored in a special format.

First we briefly introduce the format in which we store the action data, and then proceed with the method of using them in new situations.

3.3.1 Action Data Tables (ADT)

Based on the SEC framework and action definition given in Chapter 2, we have created a data format to store all the data of an executed manipulation action in a structured, re-usable and transformable way. This format is called action data table (ADT) which contains high- and low-level data for a specific example of an action performed with certain objects and poses.

Some of important fields of the ADT are summarized below:

- Instruction: The action described in a natural language sentence.
- Action name: The name of action.
- Objects: The objects in the action with their roles (main, primary, etc.) and their name, CAD models and pose.
- SEC: The SEC matrix which describes the changes of relations of object through the action.
- Action chunks: The SEC matrix divides the action into some chunks. The following information are available at each chunk:
 - The start and the end times.
 - Pose of the objects at the start and the end times.
 - The list of primitives their arguments and parameters.
- Sensor data: The recorded sensor data of each executed action is stored in a file.

```

<?xml version="1.0"?>
<action>

  <instruction>Unscrew a bottle</instruction>
  <name>Unscrew</name>
  <refframe>kuka_goe.baseframe</refframe>
  <main_object>
  <primary_object>
  <secondary_object>
  <recording_data><uri>Unscrew_bottle.bag</uri></recording_data>
  <SEC>
  <sec_line first="hand" second="main_object">
  >>   <sec_entry>N</sec_entry>
  >>   <sec_entry>T</sec_entry>
  >>   <sec_entry>T</sec_entry>
  >>   <sec_entry>N</sec_entry>
  >>   </sec_line>
  <sec_line first="main_object" second="primary_object">
  <sec_line first="main_object" second="secondary_object">
  </anchor_points>
  <action_chunks>
  >>   <action_chunk>
  >>   >>   <context>Hand moves towards lid and grasps it</context>
  >>   >>   <wrist>
  >>   >>   <main_object_act>
  >>   >>   <primary_object_act>void</primary_object_act>>>
  >>   >>   <secondary_object_act>void</secondary_object_act>>>
  >>   >>   <tool_act>void</tool_act>
  >>   >>   <grasp_ungrasp>
  >>   >>   <primitives>
  >>   </action_chunk>
  >>   <action_chunk>
  >>   <action_chunk>
  </action_chunks>
</action>

```

FIGURE 3.20: An example of a part of an ADT file for an *Unscrew* action. Actual ADTs have more entries which are omitted here to save space.

An example of an ADT is shown in 3.20. We create one ADT for each executed action and store them in a database which can be found in the following address: <http://www.acat-project.eu/index.php?page=adt>.

3.3.2 Compilation Process

Given a new instruction, we try to find the most similar action from the ontology and then fill its parameters from the ADT database. The block diagram for the compilation process is shown in Figure 3.21.

The different steps of the compilation are explained with an example. Given the scene shown on top of Figure 3.22, we wish to execute the following instruction: Shake the plastic bottle and put it on the tray.

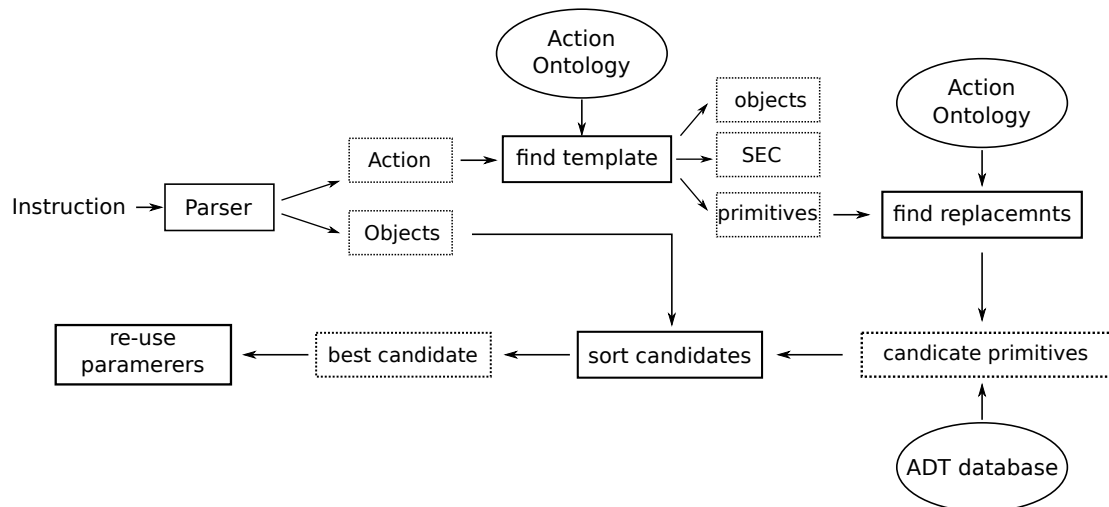


FIGURE 3.21: Block diagram of compilation process. We use two databases in this process. First is the Action ontology introduced in Section 2.2. The second is the ADT database which is the collection of our executed actions in the ADT format.



FIGURE 3.22: A scene with some objects to test the compilation algorithm. We want to execute the following instruction: Shake the plastic bottle and put it on the tray.

- Parse the instruction:

A semantic parser [65] is used to associate different parts of the instruction sentence to different components of action. For example, the verb *Shake* is extracted and used as the action name. The result of parsing of the example is shown in Table 3.4.

- Analyze the scene:

Instruction	Shake the bottle and put it on the tray
Action	Shake
main	bottle
primary	-
secondary	tray

TABLE 3.4: Output of parsing of the example instruction

We use the same vision system that we used before in Chapter 2 to recognize objects [48], find their positions and touching relations. Then we compare the recognized objects with the objects in the instruction, in the second column of Table 3.4, which come from the parser. If the parser objects are missing in the scene, action execution is not possible. The method to deal with missing objects will be presented in Chapter 4.

- Find the action template:

Here we should find the action which is similar to the action given by the parser. We restore the definition of this action from the ontology introduced in Section 2.2 and use it as the template for execution. For this example, we restore the definition of action *Shake* from the ontology, which is shown in Figure 3.23. The primitives which are shown by P_1 and P_2 at each column, should be executed. However, we still need to find their parameters from the ADT database.

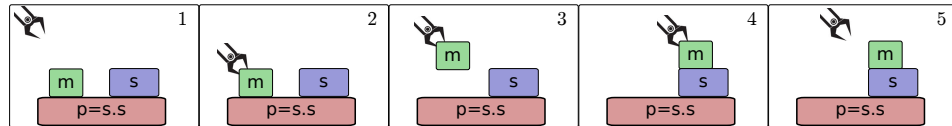
- Find the candidate replacements for each primitive:

In this step we want to determine for each primitive of the desired action (*Shake*), similar primitives from the ontology. We use the following measures to select similar primitives:

- type of primitive
- argument of primitive
- SEC context at which the primitive occurs.

Trivially, the primitives of the same action can be re-used. For example, if we want to execute the first primitive of action *Shake* which is *arm_move(main)*, we look for primitives in all actions in the ontology (See Appendix A) which have:

- *arm_move()*



	Shake				
SEC	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
primary, secondary	T	T	T	T	T
main, secondary sup.	T	T	N	N	N
Primitives					
P1	hand_preshape	arm_move (primary)	arm_move_periodic()	hand_release	
P2	arm_move (main)		arm_move (secondary)	arm_move (secondary)	
P3	hand_grasp				

FIGURE 3.23: The definition of action *Shake* which is associated to the output of parser, is fetched from the ontology. We use its SEC, abstract objects and primitives in the compilation process

- The argument is *main*
- The relation of manipulator and main changes from *N* to *T*.

The replacements of for the primitives of the *Shake* action are listed in Table.3.5.

- Find the best matching primitive:

The previous step gives the candidate primitives from which we can re-use parameters. A necessary condition for re-use is that we have an example of that action in our ADT database. These actions are marked with an asterisk in Table 3.5. Table 3.5 may give multiple candidates for a primitive. In that case we sort them to find the one most similar to the desired instruction. We use the following order to sort the candidate primitives:

1. Same action, main, primary and secondary
2. Same action, main, secondary
3. Same action, main, primary
4. Same action, main
5. Same main and secondary
6. Same main and primary

Desired Primitive			Candidate Primitive			
Action	State No.	Primitive No.	Primitive	Action	State No.	Primitive No.
Shake	1	1	<i>arm_move(main)</i>	Put on top*	1	1
Shake	1	2	<i>arm_move(main)</i>	Put on top*	1	2
Shake	2	1	<i>arm_move(primary)</i>	Put on top*	2	1
Shake	4	2	<i>arm_move(secondary)</i>	Put on top*	4	2
Shake	1	1	<i>arm_move(main)</i>	Shake*	1	1
Shake	1	2	<i>arm_move(main)</i>	Shake*	1	2
Shake	2	1	<i>arm_move(primary)</i>	Shake*	2	1
Shake	3	1	<i>arm_move_periodic()</i>	Shake*	3	1
Shake	3	2	<i>arm_move(secondary)</i>	Shake*	3	2
Shake	4	2	<i>arm_move(secondary)</i>	Shake*	4	2
Shake	1	1	<i>arm_move(main)</i>	Screw	1	1
Shake	1	2	<i>arm_move(main)</i>	Screw	1	2
Shake	2	1	<i>arm_move(primary)</i>	Screw	2	1
Shake	4	2	<i>arm_move(secondary)</i>	Screw	4	3
Shake	1	1	<i>arm_move(main)</i>	Unscrew*	1	1
Shake	1	2	<i>arm_move(main)</i>	Unscrew*	1	2
Shake	4	2	<i>arm_move(secondary)</i>	Unscrew*	4	2
Shake	1	1	<i>arm_move(main)</i>	Insert*	1	1
Shake	1	2	<i>arm_move(main)</i>	Insert*	1	2
Shake	2	1	<i>arm_move(primary)</i>	Insert*	2	1
Shake	4	2	<i>arm_move(secondary)</i>	Insert*	4	3
Shake	1	1	<i>arm_move(main)</i>	Drop*	1	1
Shake	1	2	<i>arm_move(main)</i>	Drop*	1	2
Shake	2	1	<i>arm_move(primary)</i>	Drop*	2	1
Shake	1	2	<i>arm_move(main)</i>	Align with grasp	1	2
Shake	1	1	<i>arm_move(main)</i>	Put over	1	1
Shake	1	2	<i>arm_move(main)</i>	Put over	1	2
Shake	2	1	<i>arm_move(primary)</i>	Put over	2	1
Shake	4	2	<i>arm_move(secondary)</i>	Put over	4	2
Shake	1	1	<i>arm_move(main)</i>	Lay	1	1
Shake	1	2	<i>arm_move(main)</i>	Lay	1	2
Shake	2	1	<i>arm_move(primary)</i>	Lay	2	1
Shake	4	2	<i>arm_move(secondary)</i>	Lay	4	2

TABLE 3.5: List of candidate primitives from all actions in the ontology which are similar to primitives of *Shake* action. The actions for which we have at least one example in ADT database are marked with an asterisk (*)

7. Same main
8. Same action, secondary
9. Same action, primary
10. Same action

Note that to sort the primitives which have the *Primary* (*Secondary*) object as their argument, we modify the above order to prefer the actions with similar *Primary* (*Secondary*) object.

For example, to find the parameters of the fist primitive, *hand_preshape()* of the desired action, we have the possibilities listed in Table 3.6. We have 5 possible replacements with three of them equally similar to the desired action. In the

TABLE 3.6: For the first primitive of the desired *Shake* action, the following candidates exist in the ADT database. The first three options all have the same *main* and *primary* objects.

#	Action	Main	Primary	Secondary	Similarity
1	Put on top	Bottle	Table	Cup	Main + Primary
2	Put on top	Bottle	Table	Cup	Main + Primary
3	Put on top	Bottle	Table	Jar	Main + Primary
4	Drop	Bottle	Tray	Box	Main
5	Shake	Measuring Beaker	Table	Table	Action+ Primary

execution, we can try the highest ranked parameter, and if failing, we try the other alternatives.

Similarly, we analyze the other primitives of the desired *Shake* action and find the necessary parameters from previous experiments stored in the ADT database.

3.4 Bootstrapping Results

We performed several experiments to evaluate both types of bootstrapping. First we present some results on bootstrapping from human demonstrations (Section 3.2). We analyze actions demonstrated by human operator both with kinesthetic guidance and in virtual reality and try to find their descriptive parameters. Then, we apply the algorithm presented in Section 3.3 to find execution parameters for new instructions.

3.4.1 Human Demonstration Results

We recorded 10 kinesthetically guided action and 20 simulated actions. The human demonstrations are listed in Table 3.7.

For each action the SEC matrix, object roles and action primitives are obtained. We do not evaluate the accuracy of calculating the SEC matrix, since these algorithms are

TABLE 3.7: The list of actions demonstrated by human operator. The method of demonstration, the action name and the roles of objects in the action are specified.

Demonstration Method	#	Action	main	primary	secondary
kinesthetic guidance	<i>kin</i> ₁	Put on top	Bottle	Table	Bottle Holder
	<i>kin</i> ₂	Put on top	Measuring Beaker	Table	Tray
	<i>kin</i> ₃	Put on top	Jar	Table	Bottle Holder
	<i>kin</i> ₄	Put on top	Bottle	Table	Cup
	<i>kin</i> ₅	Put on top	Jar	Table	Pot
	<i>kin</i> ₆	Put on top	Plastic Bottle	Table	Pot
	<i>kin</i> ₇	Unscrew	Bottle Lid	Bottle	Table
	<i>kin</i> ₈	Unscrew	Jar Lid	Jar	Table
	<i>kin</i> ₉	Unscrew	Bottle Lid	Bottle	Table
	<i>kin</i> ₁₀	Shake	Measuring Beaker	Table	Table
virtual reality	<i>vr</i> ₁	Drop	Pressure ring	Ring support	Mug
	<i>vr</i> ₂	Drop	Pressure ring	Ring support	Mug
	<i>vr</i> ₃	Put on top	Rotor cap	Table	Rotor Axle
	<i>vr</i> ₄	Take Down	Rotor Cap	Fixture	Table
	<i>vr</i> ₅	Take Down	Rotor Cap	Fixture	Table
	<i>vr</i> ₆	Pick and place	Rotor Axle	Shelf	Tray
	<i>vr</i> ₇	Drop	Bottle	Tray	Basket
	<i>vr</i> ₈	Drop	Rotor Cap	Table	Box
	<i>vr</i> ₉	Push apart	Bottle	Box	-
	<i>vr</i> ₁₀	Push apart	Cup	Jar	-
	<i>vr</i> ₁₁	Drop	Bottle	Shelf	Box
	<i>vr</i> ₁₂	Pick and place	Jar	Tray	Shelf
	<i>vr</i> ₁₃	Shake	Jar	Tray	Tray
	<i>vr</i> ₁₄	Drop	Bottle cap	Tray	Trash
	<i>vr</i> ₁₅	Shake	Bottle	Tray	Tray
	<i>vr</i> ₁₆	Drop	Spoon	Plate	Box
	<i>vr</i> ₁₇	Insert	Knife	Table	Jar
	<i>vr</i> ₁₈	Invert	Bottle cap	Table	Table
	<i>vr</i> ₁₉	Stir	Spoon	Table	Mug

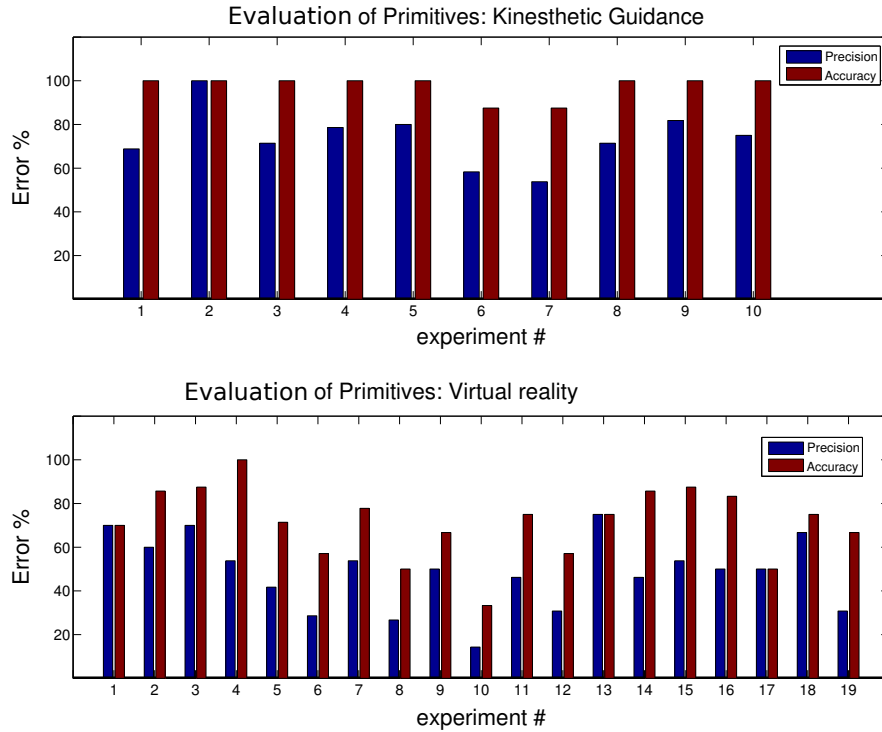


FIGURE 3.24: The results of primitive detection is evaluated by comparing the results with the human-generated ground truth. The accuracy and precision of both kinesthetically guided experiments (top) and virtual reality experiments (bottom) are shown.

not part of the current thesis. To evaluate the algorithms for finding object roles and primitives, the results are compared with the ground truth which is obtained manually. The ground truth object roles are shown in the last three columns of Table 3.7.

To evaluate the obtained primitives, we calculate the accuracy and precision of the results:

- Accuracy: Number of correct primitives divided by total number of primitives in ground truth.
- Precision: Number of correct primitives divided by number of all found primitives.

High accuracy means there are low false negatives and high precision means low false positives. The accuracy and precision for all experiments are shown in Figure 3.24. The results are impressive for kinesthetically guided experiments since in 8 out of 10 experiments the accuracy is 100%, which means that all ground truth primitives are found (top plot). The precisions are also at least 50%.

TABLE 3.8: Compilation results for the instruction *Shake the plastic bottle and put it on the tray*. For each primitive we indicate the action in the ADT database and the primitive from which the parameters are replaced.

Instruction					
state #	primitive #	primitive	Replacement		
			Action	state	primitive #
1	1	hand_preshape	<i>vr</i> ₁₉	1	1
1	2	arm_move (main)	<i>vr</i> ₁₉	1	2
2	1	arm_move (primary)	<i>kin</i> ₁₀	2	1
3	1	arm_move_periodic	<i>vr</i> ₁₉	3	1
3	2	arm_move (secondary)	<i>vr</i> ₁₉	3	2
4	2	arm_move (secondary)	<i>vr</i> ₁₉	4	2

For the virtual reality experiments, the accuracy is still high but the precision is low in some experiments. The reason for this low accuracy is that in these experiments the operator had many noisy and unnecessary movements, which introduces several false primitives and makes the detection of arguments difficult.

3.4.2 Action Compilation Experiments

The result of compilation algorithm for the *Shake the plastic bottle and put it on the tray* instruction are shown in Table C.6. To obtain these results we use our database of ADTs which contains the actions listed in Table 3.7. We also obtained the execution parameters for the following instructions and the results are shown in Appendix C.

- Push the bottle away from the jar.
- Take the jar and place it into the box.
- Drop the bottle into the wastebasket.
- Insert the spoon into the jar.
- Unscrew the lid from the mug
- Invert a jar

3.5 Conclusion

To go beyond predefined actions parameters and improve the performance of execution, we used the concept of bootstrapping to obtain execution related parameters from human demonstrations and previous experiments. We conclude that:

- The action descriptive parameters can be obtained from human demonstrations with an acceptable confidence. However, if the quality of execution is very poor, we lose precision (increased false positives). The algorithm remains quite accurate even when the quality of demonstration is not good and the operator introduces lots of noise.
- By combining the results of multiple demonstrations of the same action, even on different objects, we can filter out most of the noise of single demonstrations. This generates a near-perfect description of the action.
- The obtained description is generalizable and can be applied (within reason) to other robot platforms, objects and situations.
- To obtain the parameters of an action, we can search databases of previously executed actions provided that they are described in a compatible format.
- The parameters depend both on the action and the specific objects used. It is not trivial to detect which one should be given a higher priority in compilation. However, our proposed method generates multiple possible parameters for a primitive, and if the first one fails in execution, we can try the others.

Chapter 4

Error handling and Planning

In the previous chapters we proposed our definition of manipulation actions and presented our method of execution. Here we want to discuss the following related topics:

- Dealing with errors and failures in the execution of actions.
- Integrating manipulation actions with symbolic planning.

In Section 2.4.3 we showed the ability of our system to execute chained actions. Such chains of actions are produced by a symbolic planner. The symbolic planner generates a plan (a sequence of actions) to reach the goal states, considering the pre-conditions and the effects of the actions. To achieve this, we should define the state and actions (*planning operators*) by logical predicates.

This chapter is organized as follows: In Section 4.1 we present the details of error handling in the execution engine. Then, we define a symbolic planning domain which is compatible to the proposed ontology of actions in Section 4.2. In Section 4.3, we show cases of executing a plan even when some objects are missing. The conclusion of this chapter can be found in Section 4.4.

4.1 Fault Detection

The execution of an action can go wrong due to various problems. There could be an error in perception resulting in incorrect segments, object poses, relations or mis-recognized objects. The position or force control of the robot may be imprecise or the communication to the robot could be lost. It is also possible for the robot arm to hit a singularity or collide with itself or other objects. A major problem for the robot hand is grasping of objects which could easily fail due to its inherent complexity. Even worse, an initially successful grasp could fail at a later time due to slippage.

It could also be the case that the desired action is not afforded by the scene. For example, if there is a lid on a pot, it is not possible to put something in it.

These faults occur at different levels with different criticality. For instance a lost communication with the robot arm is a critical fault which is considered a software engineering problem. If the connection is lost, there should be a automatic mechanism to pause the execution, establish the connection again and continue the execution. Otherwise the system cannot cope with this kind of fault and the operator may need to intervene and restart manually.

Categorizing and giving solution to all possible faults in such a system is out of scope of this thesis. In general it is currently not possible to achieve full fault tolerance for robot actions outside factory floors, because situations can vary widely in unconstrained environments leading to unforeseen contingencies. However, we pay special attention to the faults which are reflected in the object relations. We show that the definition of actions based on relations between objects, provides two specific ways to *detect* errors, which are not present in other action definitions. Both of these are detectable from sensor data and action components described before.

In Section 2.3.2 we described the *Error* state in the execution engine (See Figure 2.14), which is entered under these conditions:

1. The execution of the primitives does not result in the expected change in relations
(From *Select Primitive* state)
2. The execution of the primitives causes unexpected changes in object relations
(From *Execute Primitive* state).

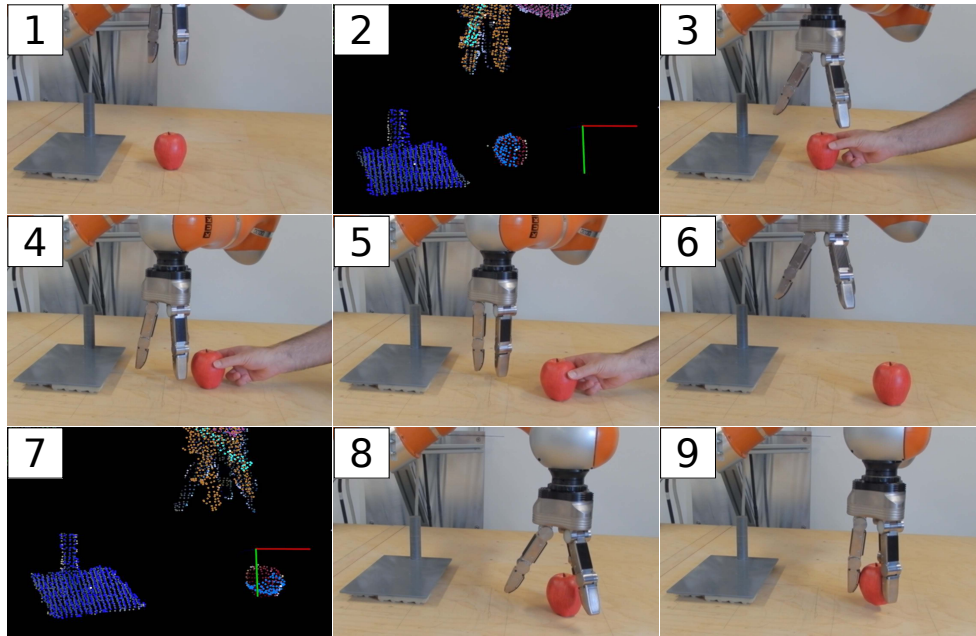


FIGURE 4.1: Error handling after failure in grasping an object. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4,5- When the manipulator approaches to grasp the apple, we move it to cause the grasp to fail. 6- The robot hand opens and the robot arm moves up waiting for a new perception. 7- New perception of the objects by vision system. 8,9- Approaching the apple in its new position and performing a grasp.

To deal with these errors, first we undo the primitives of the current state (SEC column) to reach the previous known state. Then, we evaluate the object relations again and transition to the *Check Current Relations* state and continue the execution. This results in a new perception of the position and relations of objects, by which the system decides which primitive should be executed next.

Here some examples of error handling are shown. The first two examples show errors when expected changes in relations do not happen. In Figure 4.1, the robot hand approaches the apple to grasp it, but fails, since the apple is not in the expected position. The manipulator retracts and receives the new position of the apple from the vision system and repeats the grasp. Another example is shown in Figure 4.2 during a *Put on top* action. Here, the *secondary* object (the cutting board) is displaced just before the *main* object (apple) is placed on top of it. The absence of the cutting board is detected and the manipulator retracts. After receiving the new position of the cutting board, the manipulator completes the action successfully.

An example of the second type of error, is shown in Figure 4.3,

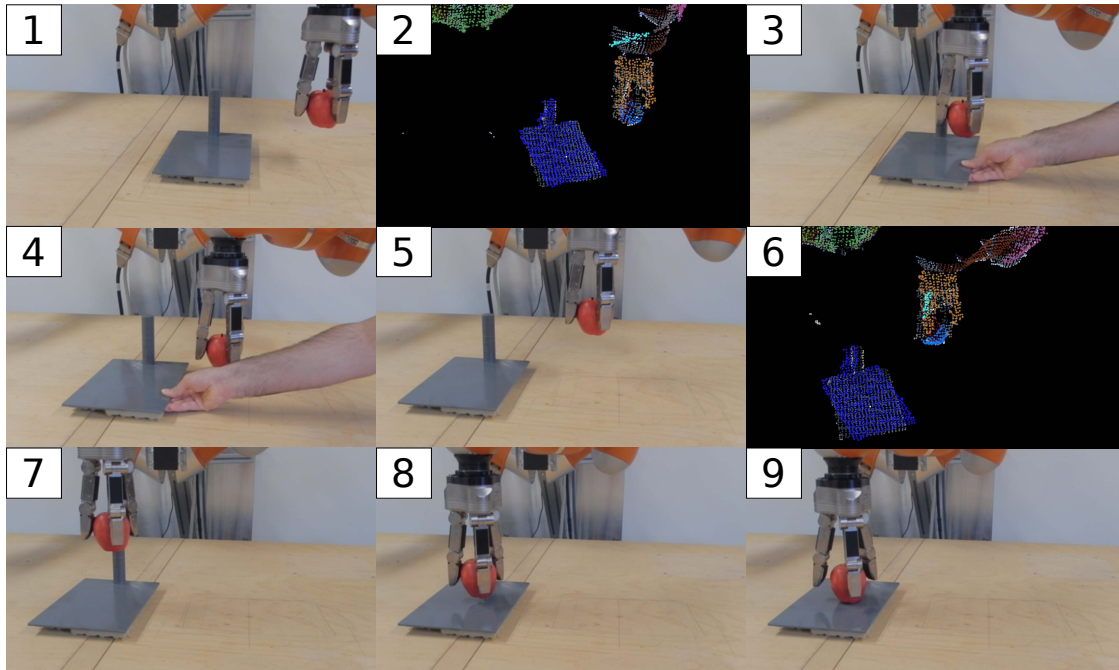


FIGURE 4.2: Error handling after failure in *Put on top* action. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4- When the manipulator tries to put the apple on top on the board, we move the board to cause the failure. 5- The manipulator moves up and waits for a new perception. 6- New perception of the objects by vision system. 7,8,9- The manipulator puts the apple on the board in the new position.

The action is *Push by holding* in which the *main* object (apple) is held from top and pushed. During the pushing, the apple slips and the action is interrupted. The error is detected and the manipulator retracts and repeats the action after receiving the new position of the apple. The second example of these errors is shown in Figure 4.4 in which after a successful grasp, the object is taken away from the robot hand. The robot detects the absence of the grasped object and reacts to it by opening the hand and moving up. After receiving the new position of the object, the grasp is repeated.

There are cases where error handling can not help, for example if we try to cut an uncuttable object (like a cup). The error handling would try to repeat cutting the cup without success. After a few unsuccessful repetitions, the system transitions to the *Failure* state.

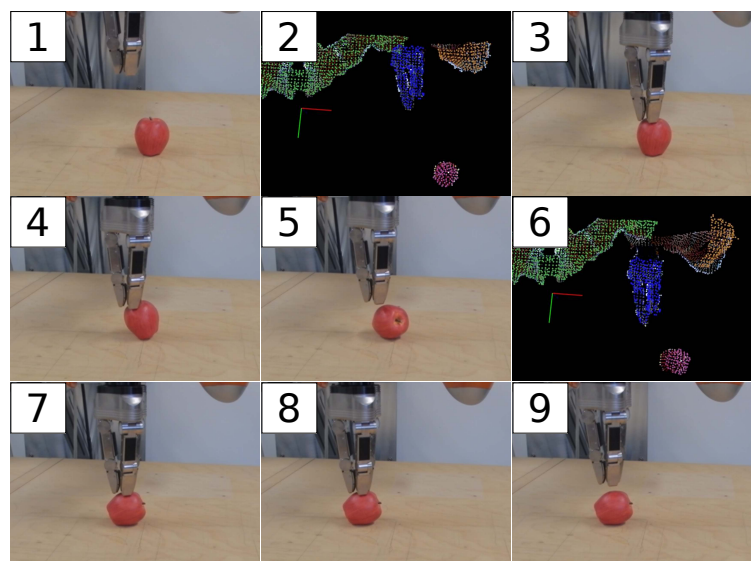


FIGURE 4.3: Error handling after failure in the push by holding action. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4,5- When the manipulator tries to hold the apple from top and push it. However, this fails due to inaccuracy in perception and the geometry of the apple. 6- The manipulator moves up and receives a new perception of the objects from the vision system. 7,8,9- The manipulator tries the pushing again and this time it succeeds.

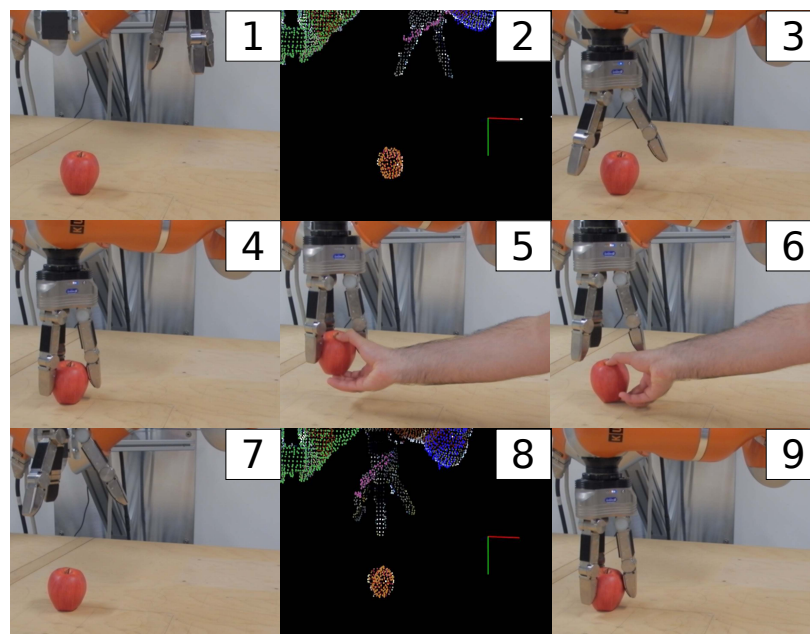


FIGURE 4.4: Error handling after the grasped objects slips through the robot hand. 1- The initial Scene. 2- Perception of the objects by vision system. 3,4- The manipulator approaches the apple and grasps it successfully. 5- The grasped object is taken out of the robot hand to cause the error. 6,7- The robot hand opens and the manipulator retracts. 8- New perception of the objects is received from the vision system. 9- The grasp is repeated successfully.

4.2 Planning with Ontology of Actions

To solve complex tasks by using the proposed execution framework, we use artificial intelligence (AI) techniques for planning and decision making.

To perform symbolic planning, we create a planning domain which is related to the manipulation actions in the ontology. This domain has the following components:

- State: The state of the environment is defined by the touching relation of objects in the scene. This means that a set of $touching(obj_1, obj_2)$ and $!touching(obj_1, obj_2)$ defines the state.
- Planning operator: For each action in the ontology, an operator is defined which transforms the state of the scene from initial to final. The initial and final states are the first and last columns of the SEC matrix of the action.

Since we defined object roles based on the touching relations, we have a general planning operator which is valid for all the actions:

operator Action(main, primary, secondary) :

Precondition :

$!touching(manipulator, main)$

$touching(main, primary)$

$!touching(main, secondary)$

Effect :

$!touching(manipulator, main)$

$!touching(main, primary)$

$touching(main, secondary)$

Individual planning operators may have additional arguments, preconditions and effects. For example, the planning operators for *Put on top* and *Take Down* actions can be

formalized as following:

operator PutOnTop(main, primary, secondary, secondary_support) :

Precondition :

!touching(manipulator, main)

touching(main, primary)

touching(primary, secondary)

touching(main, secondary_support)

!touching(main, secondary)

Effect :

!touching(manipulator, main)

!touching(main, primary)

touching(main, secondary)

touching(primary, secondary)

!touching(main, secondary_support)

operatorTakeDown(main, primary, secondary, primary_support) :

Precondition :

!touching(manipulator, main)

touching(main, primary)

touching(primary, secondary)

touching(main, primary_support)

!touching(main, secondary)

Effect :

!touching(manipulator, main)

!touching(main, primary)

touching(main, secondary)

touching(primary, secondary)

!touching(main, primary_support)

The perception calculates the relations of the objects (initial state) which is the input to the planner. The goal is provided in the same format and the planner uses the available planning operators to search for a plan. An example is shown in Figure 4.5 in which the

initial and goal states are as follows:

Initial :

!touching(apple_green, box)
touching(apple_red, box)
!touching(apple_red, table)
touching(apple_green, table)
!touching(apple_green, apple_red)
touching(box, table)

Goal :

touching(apple_green, box)
!touching(apple_red, box)
touching(apple_red, table)
!touching(apple_green, table)

If we try to plan ¹ using only the two actions *Put on top* and *Take Down*, we will get the following solutions:

- 1 – *PutOnTop(apple_green, table, box, table)*
 2 – *TakeDown(apple_red, box, table, table)*
- 1 – *TakeDown(apple_red, box, table, table)*
 2 – *PutOnTop(apple_green, table, box, table)*

Both solutions are correct given the current preconditions of the actions. However the first plan will not succeed since it tries to put the green apple on top of the box before removing the red apple. We can conclude that the preconditions which come from the SEC matrices, are not sufficient for allowing an action. As a consequence we need to learn more preconditions by integrating planning and execution which is explained in the following.

¹We use the PKS (Plan with knowledge and sensing) planner introduced in [66].



Relations	Initial	Goal
box, red ap.	T	N
box, green ap.	N	T

FIGURE 4.5: The initial and goal states for a simple planning problem.

We can learn additional preconditions by using a teacher interface. The teacher interface is part of a decision making framework developed in Agostini et al. [67]. The diagram in Figure 4.6 shows an overview of this decision making system. In this system, the planner generates a plan given the current and goal states. If it is possible to generate a plan, the robot executes it. However, if there is no possible plan or the execution fails, the teacher interface can help. The teacher (a human operator who knows the domain) solves the problem by executing the next action. Based on the observed state transitions, the systems learns the additional preconditions. Its planning operators are added or updated so that the next time, in a similar situation, a plan can be generated.

In our example, if we try to perform the *Put on top* action first, the action fails. The failure is sent to the decision making system and the teacher is asked to help. The teacher then performs the *Take down* action. A new plan can be generated now which has only one *Put on top* action. Now, since the learning module compares the state of the scene, which leads to failure, with the state of the scene in which the *Put on top* succeeds. The difference is that in the former the green apple touches the red apple and it does not touch the table. Now we can add to the preconditions of *Put on top* action that no other object should be on top of the secondary object. This way we can complete the preconditions of actions, through the teacher.

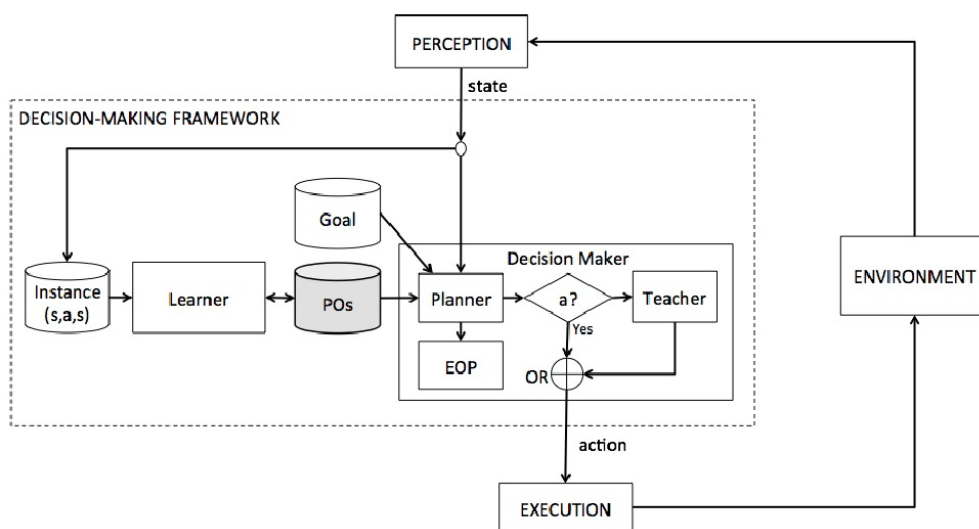


FIGURE 4.6: General diagram for plan generation and object replacement. (Reference: [1])

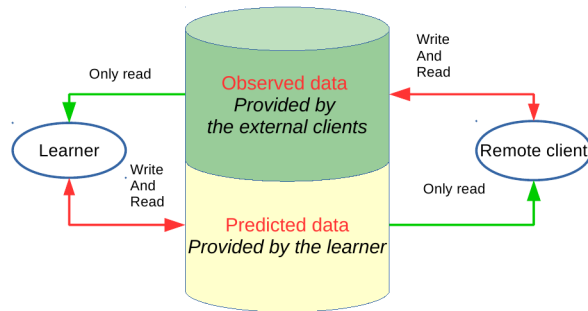


FIGURE 4.7: ROAR is an intelligent database which updates its content. (Reference: [1])

4.3 Object Replacement

In this section we demonstrate the capability of our integrated system to replace missing objects in a generated plan. This is a usual problem that is easily solved by humans provided their natural knowledge to find object substitutions: using a knife as a screwdriver or a book as a cutting board. On the other hand, in robotic applications, objects required in the task should be included in advance in the problem definition. If any of these objects is missing from the scenario, the conventional approach is to manually redefine the problem according to the available objects in the scene.

Our approach, presented in detail in [1], uses a logic-based planner to generate a plan from a prototypical problem definition and searches for replacements in the scene when some of the objects involved in the plan are missing.

This is done by means of a repository of objects and attributes with roles (ROAR) [68], which is used to identify the affordances of the unknown objects in the scene. The structure of ROAR database is shown in Figure 4.7. Here we discuss the following planning problems, related to the execution framework.

- Replace Cucumber: Here we use the same salad making problem which is shown in Figure 2.23. In the original planning problem, a cucumber is cut and moved to a bowl. In the scene shown in Figure 4.8, there is no cucumber. However there are two additional objects: The cup and the banana.

The cucumber is used in several actions in the original plan: *Put on top*, *Cut* and *Unload*. We make queries to ROAR about the jar and the banana, to see which of them can play the role of a cucumber. The ROAR detects that the banana may



FIGURE 4.8: The modified scene for salad making in which the cucumber is missing. There are two additional objects, the jar and the banana.

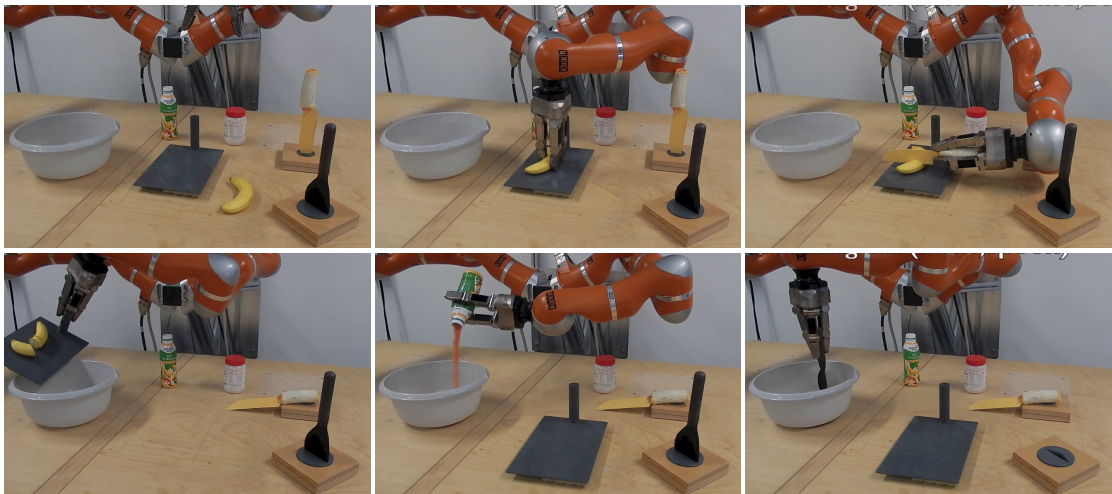


FIGURE 4.9: The modified salad making which uses the banana instead of cucumber.

replace the cucumber, while the jar is not proper for this. The plan is updated and sent for execution. The result is shown in Figure 4.9.

- It is also possible to replace tools. In Figure 4.10 we show execution of a plan with two actions: unload and stir. The stir is done by using a spoon. In the scene in Figure 4.11 we have a similar scene without the spoon. However, there are a knife and a bottle in the scene which we query ROAR to check their affordances. ROAR confirms that a knife can also be used for stirring, therefore we can execute stirring by a knife.



FIGURE 4.10: The execution of chained actions: Unload and Stir by using the spoon.



FIGURE 4.11: The replacement of the spoon by the knife. The plan is updated to use knife instead of spoon.

4.4 Conclusion

In this chapter we tried to explore error handling and integration of our execution framework with symbolic planners. The error handling can deal with random errors which happen in the execution by repeating the action from a trusted point after receiving a new percept of the system. The systematic errors, however, can not be dealt with this way and after a few unsuccessful repetitions we declare the failure of the execution. These errors are either impossible to solve or they need intervention of a higher level entity in the system (either an operator or a planner) to change the strategy.

Chapter 5

Short Summary and Final Remarks

This chapter concludes this thesis with a short summary and final remarks. Here we highlight the most important points of the thesis. The summary of each chapter is already provided in their conclusion sections.

5.1 Summary

This work is based on previous research on manipulation actions especially the SEC framework [14] and ontology of actions [45]. The main goal is to provide means of executing a range of human-like manipulation actions using a arm-hand robot system.

First we provided a layered definition of manipulation actions with both high-level and low-level components. To enable the execution, we proposed a mid-layer and implemented the whole framework in an integrated ROS-OROCOS software framework.

The quality of execution is improved by using parameters of actions obtained from human demonstrations. To further enable the robot to execute new instructions we designed a process to search previous experiments and find proper parameters which are applicable in the current situation.

The execution framework is integrated with a symbolic planner so that both can benefit. The execution receives a plan (chain of actions) from the planner which means more

complex tasks can be performed. The execution also provides feedback to the planner and raises errors when something goes wrong in the execution. The execution framework does not depend on the specific object which enables the planner to even update the plan in case of missing object.

5.2 Discussion of Results

The different methods are applied on a real robot arm-hand system to evaluate the performance. Several experiments on single and chained actions show that the system is able to perform a variety of actions. The single actions have success rates from 30% to 100% depending on the type of actions.

The analysis of human demonstrations show promising results especially in the kinesthetically guided experiments. The obtained primitives match the ground truth primitives in more than 70% of the times. The results are not as good for the virtual reality demonstrations, due to excessive noise in the demonstrations. Combination of multiple demonstrations of a single action, however, can remove most of this noisy results and generate a good description of actions.

5.3 Future Work

Based on the experience gained in this work the following paths are suggested to improve the results:

- We can improve the description of actions by adding more relations other than touching/not-touching relations. However, this should be done with care not to lose the advantages of the SEC framework. We can add simple relations like *on()*, *in()* and *around()* provided that the perception can consistently detect them.
- We can use approaches similar to motor babbling at the level of action primitives to find more about the state transitions induced by performing sequence of random primitives.
- The extension of the current ontology to bi-manual tasks enables us to perform more/better actions while introducing new challenges.

Appendix A

Ontology of Actions

The ontology of actions is summarized in Table A.1. The categories, sub-categories and actions are as follows:

- **Actions with main support:**
- **Actions with hand, main and main support:**

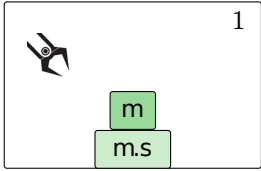
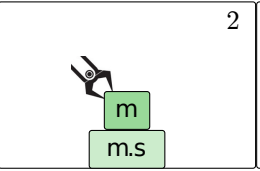
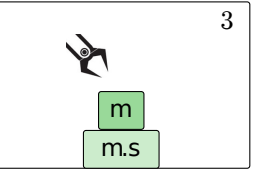
Action Name: Poke, Press

Relation / State	1	2	3
manipulator, main	N	T	N
main, main support	T	T	T
Abstract Primitives			
P_1	hand_preshape	arm_move(main)	
P_2	arm_move(main)	-	

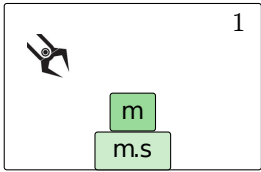
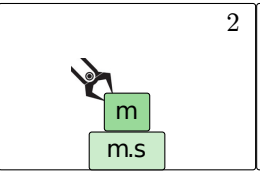
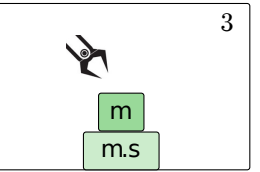
Category	Sub-Category	Example Actions
Actions with main support	Actions with hand, main and main support	push, punch, flick
	Actions with hand, main, main support and primary	push apart, cut, chop
	Actions with hand, main, main support and secondary	push together
	Actions with hand, main, main support, primary and secondary	push from a to b
Actions without main support.	primary \neq secondary and primary support \neq secondary support	pick and place, break off
	primary \neq secondary and primary support = secondary support	pick and place, break off
	primary \neq secondary and primary = secondary support	put on top
	primary \neq secondary and primary support = secondary	pick apart
	primary = secondary	pick and place, break off
Actions with load and container	The relation of load and main changes from N to T (loading)	Pipetting
	The relation of load and main changes from T to N (unloading)	Pour, Drop

TABLE A.1: Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category.

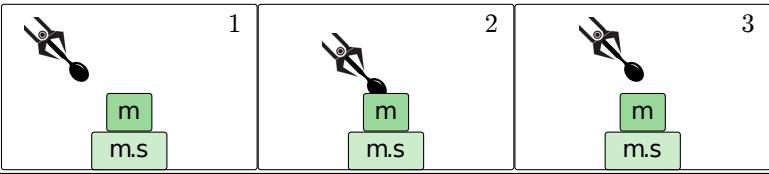
Action Name: Push

	1	2	3
			
Relation / State	1	2	3
manipulator, main	N	T	N
main, main support	T	T	T
Abstract Primitives			
P_1	hand_preshape	arm_move(main)	
P_2	arm_move(main)	arm_move(main)	

Action Name: Rotate

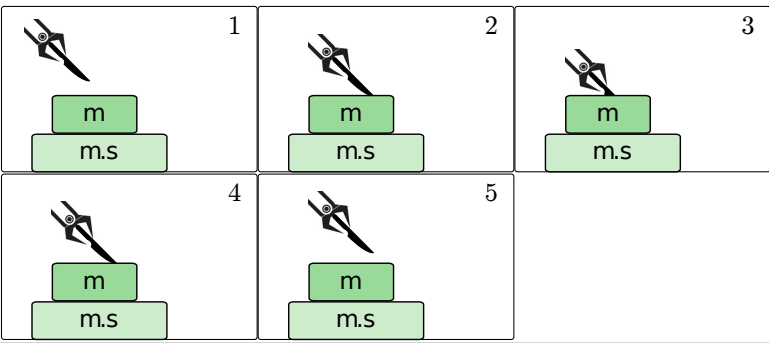
	1	2	3
			
Relation / State	1	2	3
manipulator, main	N	T	N
main, main support	T	T	T
Abstract Primitives			
P_1	hand_preshape	arm_rotate()	
P_2	arm_move(main)	hand_release()	
P_3	hand_grasp()	arm_move(free)	

Action Name: Stir (using a tool)



Relation / State	1	2	3
manipulator, tool	T	T	T
tool, main	N	T	N
main, main support	T	T	T
Abstract Primitives			
P_1	arm_move(main)	arm_periodic()	
P_2	-	hand_move(main)	

Action Name: Cut (using a tool)



Relation / State	1	2	3	4	5
manipulator, tool	T	T	T	T	T
tool, main	N	T	T	T	N
main, main support	T	T	T	T	T
tool, main support	N	N	T	N	N
Abstract Primitives					
P_1	arm_move (main)	arm_periodic()	arm_move (main_sup.)	arm_move (main)	
P_2	-	arm_exert()	-	-	

• **Actions with hand, main, main support and primary:**

Action Name: Push apart

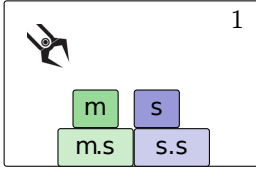
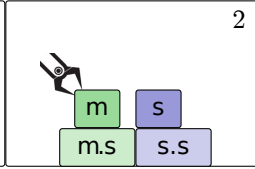
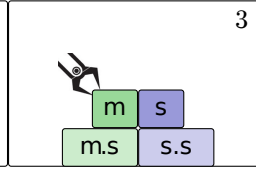
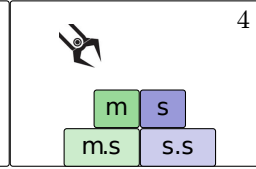
Relation / State	1	2	3	4
manipulator, main	N	T	T	N
main, primary	T	T	N	N
main, main support	T	T	T	T
primary, primary support	T	T	T	T
Abstract Primitives				
P_1	hand_preshape ()	arm_move (primary)	arm_move (main)	
P_2	arm_move (main)	-	-	

Action Name: Cut away (using a tool)

Relation / State	1	2	3	4	5
manipulator, tool	T	T	T	T	T
tool, main	N	T	T	T	N
main, main support	T	T	T	T	T
tool, main support	N	N	T	N	N
main, primary	A	A	T	N	N
Abstract Primitives					
P_1	arm_move (main)	arm_periodic()	arm_move (primary)	arm_move (main)	
P_2	-	arm_exert()	-	-	

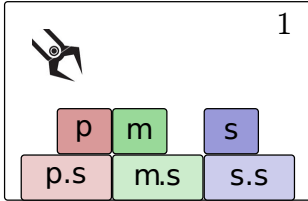
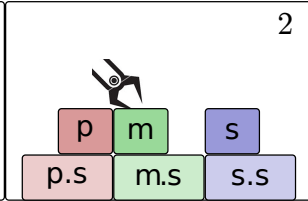
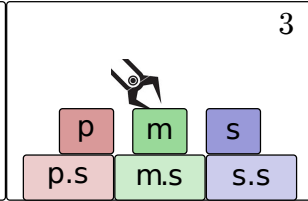
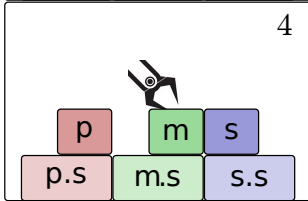
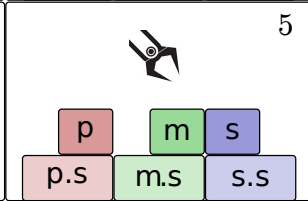
- Actions with hand, main, main support and secondary:

Action Name: Push Together

				
Relation / State	1	2	3	4
manipulator, main	N	T	T	N
main, secondary	N	N	T	T
main, main support	T	T	T	T
secondary, secondary sup.	T	T	T	T
Abstract Primitives				
P_1	hand_preshape ()	arm_move (secondary)	arm_move (main)	
P_2	arm_move (main)	-	-	

- Actions with hand, main, main support, primary and secondary:

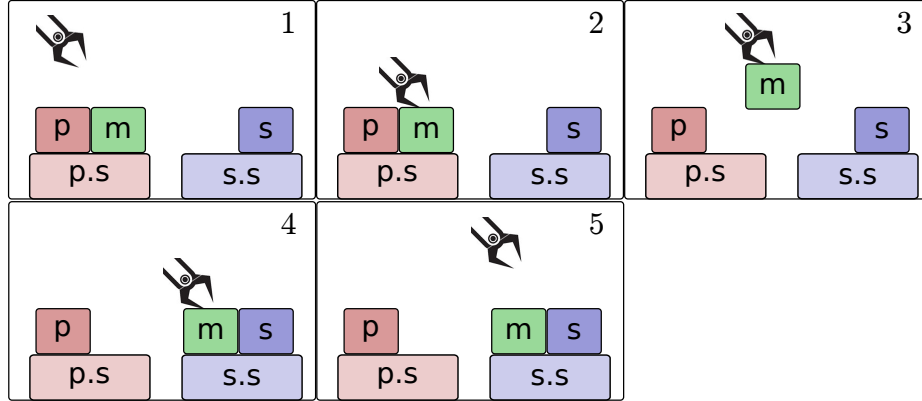
Action Name: Push From To

					
					
Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	T	T	N
main, main support	T	T	T	T	T
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	arm_move (main)	
P_2	arm_move (main)	-	-	-	

- **Actions without main support:**

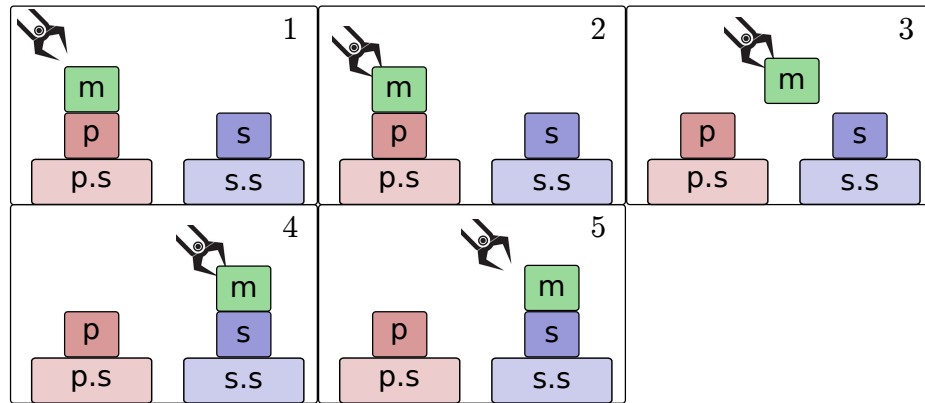
- **primary \neq secondary and primary support \neq secondary support:**

Action Name: Pick and place (side to side)



Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
main, primary support	T	T	N	N	N
main, secondary support	N	N	N	T	T
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	arm_move (secondary)	
P_3	hand_grasp ()	-	-	-	

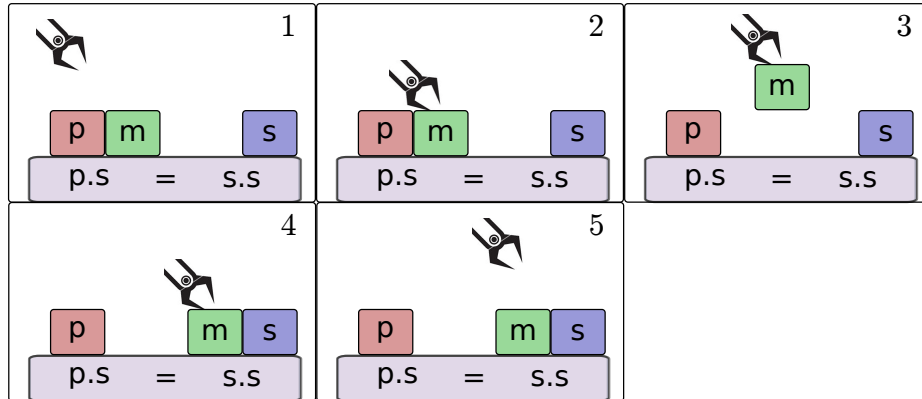
Action Name: Pick and place (top to top)



Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
main, primary support	N	N	N	N	N
main, secondary support	N	N	N	N	N
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	arm_move (secondary)	
P_3	hand_grasp ()	-	-	-	

- **primary \neq secondary and primary support = secondary support:**

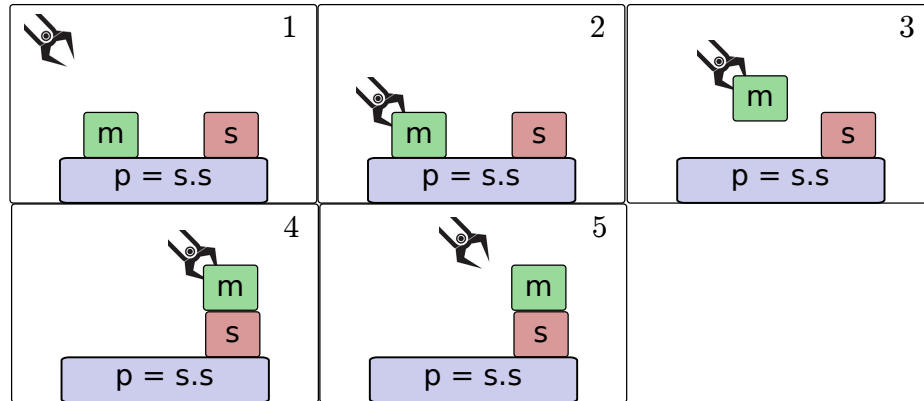
Action Name: Pick and place (side to side)



Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
main, primary support	T	T	N	T	T
main, secondary support	T	T	N	N	N
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	arm_move (secondary)	
P_3	hand_grasp ()	-	-	-	

- primary \neq secondary and primary = secondary support:

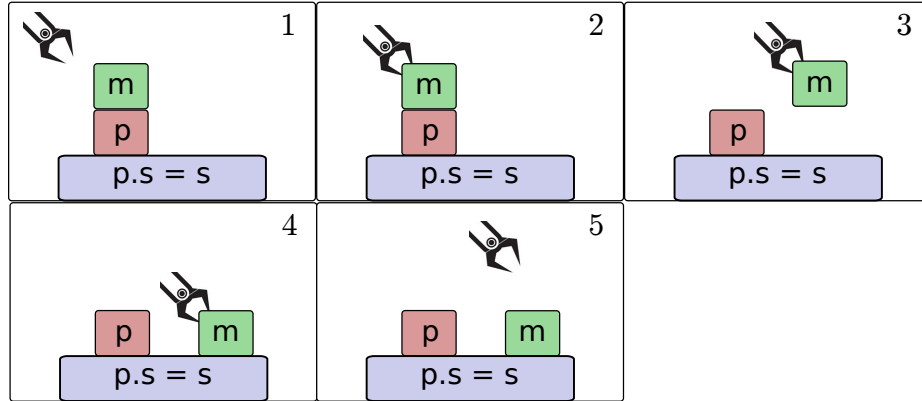
Action Name: Put on top



Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
main, secondary support	T	T	N	T	T
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	arm_move (secondary)	
P_3	hand_grasp ()	-	-	-	

- primary \neq secondary and primary support = secondary:

Action Name: Take Down

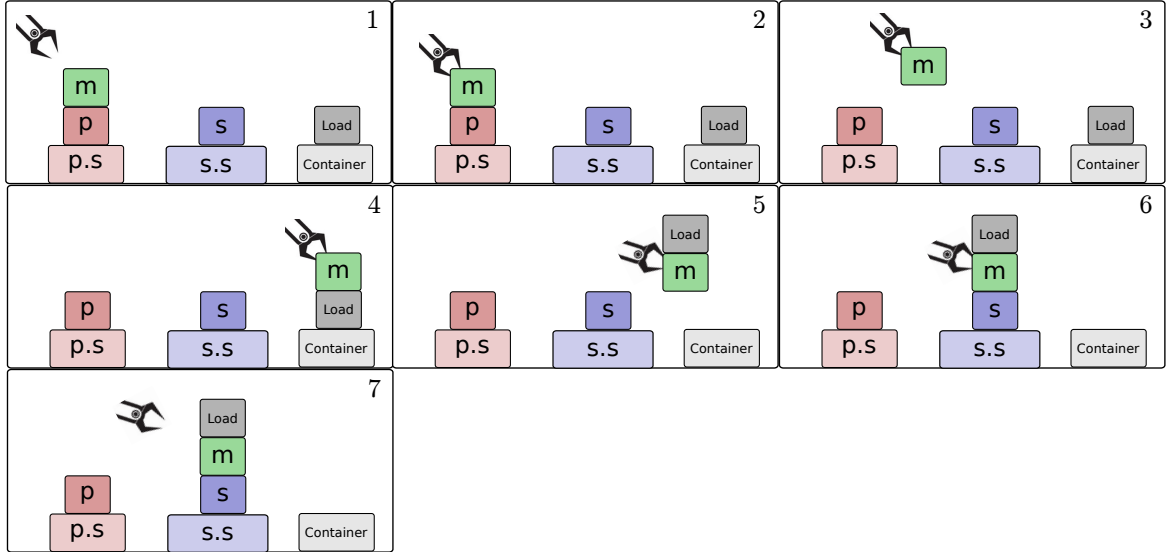


Relation / State	1	2	3	4	5
manipulator, main	N	T	T	T	N
main, primary	T	T	N	N	N
main, secondary	N	N	N	T	T
main, primary support	N	N	N	T	T
Abstract Primitives					
P_1	hand_preshape ()	arm_move (primary)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	arm_move (secondary)	
P_3	hand_grasp ()	-	-	-	

- **Actions with load and container:**

- **The relation of load and main changes from N to T (loading):**

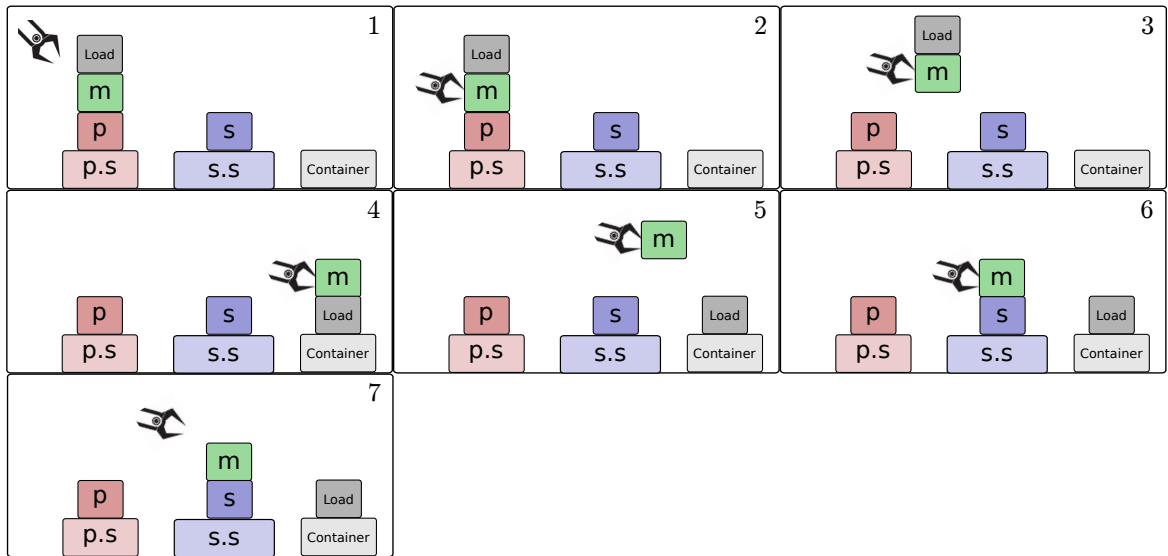
Action Name: Load



Relation / State	1	2	3	4	5	6	7
manipulator, main	N	T	T	T	N	T	N
main, primary	T	T	N	N	N	T	N
main, secondary	N	N	N	N	N	T	T
main, load	N	N	N	T	T	T	T
load, container	T	T	T	T	N	N	N
Abstract Primitives							
P_1	hand_preshape ()	arm_move (primary)	arm_move (load)	arm_move (container)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	-	-	arm_move (secondary)	
P_3	hand_grasp()	-	-	-	-	-	

- The relation of load and main changes from T to N (unloading):

Action Name: Unload, Pour



Relation / State	1	2	3	4	5	6	7
manipulator, main	N	T	T	T	N	T	N
main, primary	T	T	N	N	N	T	N
main, secondary	N	N	N	N	N	T	T
main, load	T	T	T	T	N	N	N
load, container	N	N	N	T	T	T	T
Abstract Primitives							
P_1	hand_preshape ()	arm_move (primary)	arm_move (container)	arm_move (load)	arm_move (secondary)	hand_release ()	
P_2	arm_move (main)	-	-	-	-	arm_move (secondary)	
P_3	hand_grasp()	-	-	-	-	-	

Appendix B

Execution of Single Actions

The detail of single-action experiments are presented here.

TABLE B.1: Pick and place

#	main category	main object	Results			success rate %	failure rate %
			success	failure	total		
1	round fruit	red apple	3	0	3	100	0
2	cup	yellow cup	3	0	3	100	0
3	cup	blue cup	3	0	3	100	0
4	round fruit	orange	3	0	3	100	0
5	container	orange bucket	0	3	3	0	100
6	cube	box	3	0	3	100	0
7	long fruit	cucumber	3	0	3	100	0
8	long fruit	eggplant	3	0	3	100	0
9	plate	plate	0	3	3	0	100
10	cube	small cube	2	1	3	66.7	33.3
Total			23	7	30	76.7	23.3

TABLE B.2: Push with Grasp

#	main category	main object	Results			success rate %	failure rate %
			success	failure	total		
1	round fruit	apple	3	0	3	100	0
2	cup	Blue cup	3	0	3	100	0
3	cube	Small cube	3	0	3	100	0
4	cube	box	3	0	3	100	0
5	round fruit	orange	3	0	3	100	0
6	long fruit	banana	3	0	3	100	0
7	cup	yellow cup	3	0	3	100	0
8	round fruit	red apple	3	0	3	100	0
9	long fruit	zucchini	3	0	3	100	0
10	long fruit	eggplant	3	0	3	100	0
Total			30	0	30	100	0

TABLE B.3: Push by Holding

#	main category	main object	Results		total	success rate %	failure rate %
			success	failure			
1	cube	box	3	0	3	100	0
2	round fruit	green apple	3	0	3	100	0
3	long fruit	zucchini	1	2	3	33.3	66.7
4	long fruit	eggplant	0	3	3	0	100
5	long fruit	banana	1	2	3	33.3	66.7
6	cup	yellow cup	0	3	3	0	100
7	round fruit	orange	1	2	3	33.3	66.7
8	cube	small cube	3	0	3	100	0
9	plate	green plate	0	3	3	0	100
10	cup	blue cup	0	3	3	0	100
Total			12	18	30	40	60

TABLE B.4: Poke

#	main category	main object	Results		total	success rate %	failure rate %
			success	failure			
1	cube	box	3	0	3	100	0
2	round fruit	apple	3	0	3	100	0
3	cup	blue cup	0	3	3	0	100
4	long fruit	zucchini	3	0	3	100	0
5	long fruit	eggplant	3	0	3	100	0
6	cup	yellow cup	0	3	3	0	100
7	round fruit	orange	3	0	3	100	0
8	cube	small cube	3	0	3	100	0
9	plate	gray board	3	0	3	100	0
10	plate	green plate	0	3	3	0	100
Total			21	9	30	70	30

TABLE B.5: Put on Top

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	success	failure
					success	failure			
1	round fruit	red apple	cube	box	3	0	3	100	0
2	cup	blue cup	container	bucket	3	0	3	100	0
3	round fruit	orange	container	bucket	3	0	3	100	0
4	round fruit	apple	cup	yellow cup	3	0	3	100	0
5	round fruit	apple	plate	plate	3	0	3	100	0
6	round fruit	orange	plate	plate	3	0	3	100	0
7	cup	blue cup	plate	plate	3	0	3	100	0
8	cube	small cube	plate	plate	0	3	3	0	100
9	long fruit	cucumber	plate	gray board	3	0	3	100	0
10	cup	blue cup	plate	gray board	3	0	3	100	0
Total					27	3	30	90	10

TABLE B.6: Take Down

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	Rate	
					success	failure		success	failure
1	round fruit	red apple	cube	box	3	0	3	100	0
2	cup	blue cup	container	bucket	3	0	3	100	0
3	round fruit	orange	container	bucket	3	0	3	100	0
4	round fruit	apple	cup	yellow cup	3	0	3	100	0
5	round fruit	apple	plate	plate	0	3	3	0	100
6	round fruit	orange	plate	gray board	3	0	3	100	0
7	cup	blue cup	plate	gray board	3	0	3	100	0
8	cube	small cube	cup	yellow cup	3	0	3	100	0
9	long fruit	banana	plate	gray board	3	0	3	100	0
10	cup	blue cup	cube	box	3	0	3	100	0
Total					27	3	30	90	10

TABLE B.7: Push apart by Holding

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	Rate	
					success	failure		success	failure
1	cube	box	round fruit	green apple	3	0	3	100	0
2	round fruit	green apple	cube	box	3	0	3	100	0
3	long fruit	zucchini	cube	box	1	2	3	33.3	66.7
4	long fruit	eggplant	cube	box	0	3	3	0	100
5	long fruit	banana	cube	box	1	2	3	33.3	66.7
6	cup	yellow cup	cube	box	0	3	3	0	100
7	round fruit	orange	cube	box	0	3	3	0	100
8	cube	small cube	round fruit	green apple	3	0	3	100	0
9	plate	green plate	cube	box	0	3	3	0	100
10	cup	blue cup	cube	box	0	3	3	0	100
Total					11	19	30	36.7	63.3

TABLE B.8: Push together by Holding

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	Rate	
					success	failure		success	failure
1	cube	box	round fruit	green apple	3	0	3	100	0
2	round fruit	green apple	cube	box	3	0	3	100	0
3	long fruit	zucchini	cube	box	1	2	3	33.3	66.7
4	long fruit	eggplant	cube	box	0	3	3	0	100
5	long fruit	banana	cube	box	1	2	3	33.3	66.7
6	cup	yellow cup	cube	box	0	3	3	0	100
7	round fruit	orange	cube	box	0	3	3	0	100
8	cube	small cube	round fruit	green apple	3	0	3	100	0
9	plate	green plate	cube	box	0	3	3	0	100
10	cup	blue cup	cube	box	0	3	3	0	100
Total					11	19	30	36.7	63.3

TABLE B.9: Cutting

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	Rate	
					success	failure		success	failure
1	knife	yellow knife	long fruit	cucumber	3	0	3	100	0
2	knife	yellow knife	long fruit	zucchini	3	0	3	100	0
3	knife	yellow knife	long fruit	carrot	0	3	3	0	100
4	knife	yellow knife	long fruit	banana	2	1	3	66.7	33.3
5	knife	yellow knife	long fruit	eggplant	0	3	3	0	100
6	knife	yellow knife	round fruit	apple	0	3	3	0	100
7	knife	yellow knife	round fruit	orange	0	3	3	0	100
8	knife	red knife	long fruit	cucumber	3	0	3	100	0
9	knife	red knife	long fruit	zucchini	3	0	3	100	0
10	knife	red knife	long fruit	banana	2	1	3	66.7	33.3
Total					16	14	30	54	46

TABLE B.10: Stirring

#	main category	main object	primary/ secondary category	primary/ secondary object	Results		total	Rate	
					success	failure		success	failure
1	spoon	black spoon	cup	blue cup	0	3	3	0	100
2	spoon	black spoon	cup	yellow cup	0	3	3	0	100
3	spoon	black spoon	container	orange bucket	3	0	3	100	0
4	spoon	black spoon	container	white bowl	3	0	3	100	0
5	spoon	black spoon	plate	green plate	0	3	3	0	100
6	spoon	orange spoon	cup	blue cup	0	3	3	0	100
7	spoon	orange spoon	cup	yellow cup	3	0	3	100	0
8	spoon	orange spoon	container	orange bucket	3	0	3	100	0
9	spoon	orange spoon	container	white bowl	3	0	3	100	0
10	spoon	orange spoon	plate	green plate	0	3	3	0	100
Total					15	15	30	50	50

Appendix C

Compilation Results

Here the compilation results of 10 instructions are presented.

TABLE C.1: Instruction: *Push the bottle away from the jar.*

Instruction					
state #	primitive #	primitive	Replacement		
			Action	state	primitive #
1	1	arm_move (main)	vr_9	1	1
2	1	arm_move (primary)	vr_{10}	2	1
3	1	arm_move (main)	vr_9	3	1

TABLE C.2: Instruction: *Take the jar and place it into the box.*

Instruction					
state #	primitive #	primitive	Replacement		
			Action	state	primitive #
1	1	hand_preshape	kin_5	1	1
1	2	arm_move (main)	kin_5	1	2
2	1	arm_move (primary)	kin_5	2	1
3	1	arm_move (secondary)	kin_5	3	1
4	2	arm_move (secondary)	kin_5	4	2

TABLE C.3: Instruction: *Drop the bottle into the wastebasket.*

Instruction					
state #	primitive #	primitive	Replacement		
			Action	state	primitive #
1	1	hand_preshape	vr_7	1	1
1	2	arm_move (main)	vr_7	1	2
2	1	arm_move (primary)	vr_8	2	1
3	1	arm_move (secondary)	vr_{14}	3	1

TABLE C.4: Instruction: *Insert the spoon into the jar.*

Instruction					
			Replacement		
state #	primitive #	primitive	Action	state	primitive #
1	1	hand_preshape	vr_{16}	1	1
1	2	arm_move (main)	vr_{16}	1	2
2	1	arm_move (primary)	vr_{17}	2	1
3	1	arm_move (secondary)	vr_{17}	3	1
4	3	arm_move (secondary)	vr_{17}	4	3

TABLE C.5: Instruction: *Unscrew the lid from the mug.*

Instruction					
			Replacement		
state #	primitive #	primitive	Action	state	primitive #
1	1	hand_preshape	kin_9	1	1
1	2	arm_move (main)	kin_9	1	2
2	1	arm_rotate	kin_9	2	1
2	2	arm_move (primary)	kin_8	2	2
3	1	arm_move (secondary)	kin_8	3	1
4	2	arm_move (secondary)	kin_8	4	2

TABLE C.6: Instruction: *Invert a jar*

Instruction					
			Replacement		
state #	primitive #	primitive	Action	state	primitive #
1	1	hand_preshape	kin_5	1	1
1	2	arm_move (main)	kin_5	1	2
2	1	arm_move (primary)	vr_{18}	2	1
3	1	arm_rotate	vr_{18}	3	1
3	2	arm_move (secondary)	vr_{18}	3	2
4	2	arm_move (secondary)	vr_{18}	4	2

Bibliography

- [1] A. Agostini, M. J. Aein, S. Szedmak, E. E. Aksoy, J. Piater, and F. Wörgötter. Using structural bootstrapping for object substitution in robotic executions of human-like manipulation tasks. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2015)*, pages 6479–6486, 09 2015.
- [2] J. A. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. 2002 IEEE Int. Conf. Robotics and Automation*, pages 1398–1403, 2002.
- [3] Aleš Ude. Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, 11(2):113 – 127, 1993. ISSN 0921-8890. doi: 10.1016/0921-8890(93)90015-5. URL <http://www.sciencedirect.com/science/article/pii/0921889093900155>.
- [4] D. Lee and Y. Nakamura. Stochastic model of imitating a new observed motion based on the acquired motion primitives. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4994 –5000, oct. 2006. doi: 10.1109/IROS.2006.282525.
- [5] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):286 –298, april 2007. ISSN 1083-4419. doi: 10.1109/TSMCB.2006.886952.
- [6] Katerina Pastra and Yiannis Aloimonos. The minimalist grammar of action. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 367(1585):103–117, 2012.

-
- [7] Rüdiger Dillmann, Tamim Asfour, Martin Do, Rainer Jäkel, Alexander Kasper, Pedram Azad, Aleš Ude, Sven Schmidt-Rohr, and Martin Lösch. Advances in robot programming by demonstration. *KI - Künstliche Intelligenz*, 24:295–303, 2010. ISSN 0933-1875. URL <http://dx.doi.org/10.1007/s13218-010-0060-0>. 10.1007/s13218-010-0060-0.
- [8] Dongheui Lee and Christian Ott. Incremental kinesthetic teaching of motion primitives using the motion refinement tube. *Autonomous Robots*, 31(2-3):115–131, 2011.
- [9] Tobias Luksch, Michael Gienger, Manuel Mühlig, and Takahide Yoshiike. A dynamical systems approach to adaptive sequencing of movement primitives. In *Proceedings of the 7th German Conference on Robotics (Robotik 2012)*, 2012. to be published.
- [10] M. Pardowitz, S. Knoop, R. Dillmann, and R.D. Zollner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):322–332, april 2007. ISSN 1083-4419. doi: 10.1109/TSMCB.2006.886951.
- [11] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358–363, sept. 2006. doi: 10.1109/ROMAN.2006.314460.
- [12] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.
- [13] Kyuhwa Lee, Yanyu Su, Tae-Kyun Kim, and Yiannis Demiris. A syntactic approach to robot imitation learning using probabilistic activity grammars. *Robotics and Autonomous Systems*, 61(12):1323–1334, December 2013.
- [14] E. E. Aksoy, A. Abramov, F. Wörgötter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 398–405, may 2010.
- [15] E. E. Aksoy, B. Dellen, M. Tamosiunaite, and F. Wörgötter. Execution of a dual-object (pushing) action with semantic event chains. In *IEEE-RAS International Conference on Humanoid Robots*, pages 576–583, 2011.

-
- [16] E. E. Aksoy, M. Tamosiunaite, R. Vuga, A. Ude, C. Geib, M. Steedman, and F. Wörgötter. Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots. In *IEEE Int. Conf. on Development and Learning and Epigenetic Robotics*, 2013.
- [17] E. E. Aksoy, M. Tamosiunaite, and F. Wörgötter. Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems (RAS)*, 2014.
- [18] MJ Aein, EE Aksoy, M Tamosiunaite, J Papon, A Ude, and F Worgotter. Toward a library of manipulation actions based on semantic object-action relations. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4555–4562. IEEE, 2013.
- [19] M. Wächter, S. Schulz, T. Asfour, E. E. Aksoy, F. Wörgötter, and R. Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *IEEE/RAS Int. Conf. on Humanoid Robots*, 2013.
- [20] Guoliang Luo, N. Bergstrom, C.H. Ek, and D. Kragic. Representing actions with kernels. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2028–2035, 2011.
- [21] Rok Vuga, Eren Erdal Aksoy, Florentin Wörgötter, and Ales Ude. Probabilistic semantic models for manipulation action representation and extraction. In *Robotics and Autonomous Systems (RAS)*, 2014.
- [22] David Martinez, Guillem Alenya, Pablo Jimenez, Carme Torras, Jürgen Rossmann, Nils Wantia, Eren Erdal Aksoy, Simon Haller, and Justus Piater. Active learning of manipulation sequences. In *IEEE Int. Conf. on Robotics and Automation*, 2014.
- [23] Yezhou Yang, Cornelia Fermuller, and Yiannis Aloimonos. Detection of manipulation action consequences (MAC). *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2563–2570, 2013. ISSN 10636919. doi: 10.1109/CVPR.2013.331.
- [24] Seyed Reza Ahmadzadeh, Ali Paikan, Fulvio Mastrogiovanni, Lorenzo Natale, Petar Kormushev, and Darwin G Caldwell. Learning symbolic representations of actions from human demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3801–3808. IEEE, 2015.

- [25] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Constructing symbolic representations for high-level planning. 2014.
- [26] Shuonan Dong and Brian Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *International Journal of Social Robotics*, 4(4):357–368, 2012.
- [27] Volker Krüger, Danica Kragic, Aleš Ude, and Christopher Geib. The meaning of action: a review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007.
- [28] Akihiko Yamaguchi, Christopher G Atkeson, Scott Niekum, and Tsukasa Ogasawara. Learning pouring skills from demonstration and practice. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 908–915. IEEE, 2014.
- [29] S. Morante, J.G. Victores, A. Jardon, and C. Balaguer. On using guided motor primitives to execute continuous goal-directed actions. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, pages 613–618, Aug 2014. doi: 10.1109/ROMAN.2014.6926320.
- [30] Sanjeev Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stephen Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 639–646. IEEE, 2014.
- [31] Michael Beetz, M Lorenz, and Moritz Tenorth. C RAM — A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. pages 1012–1017, 2010.
- [32] Bojan Nemeč and Aleš Ude. Action sequencing using dynamic movement primitives. *Robotica*, 30(05):837–846, 2012.
- [33] S Muench, J Kreuziger, M Kaiser, and R Dillman. Robot programming by demonstration (rpd)-using machine learning and user interaction methods for the development of easy and comfortable robot programming systems. In *Proceedings of the International Symposium on Industrial Robots*, volume 25, pages 685–685. INTERNATIONAL FEDERATION OF ROBOTICS, & ROBOTIC INDUSTRIES, 1994.

- [34] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 241–248. ACM, 2003.
- [35] R Zoliner, Michael Pardowitz, Steffen Knoop, and Rüdiger Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *Proceedings of the 2005 IEEE International Conference On Robotics and Automation*, pages 1535–1540. IEEE, 2005.
- [36] Joe Saunders, Chrystopher L Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 118–125. ACM, 2006.
- [37] Michael Pardowitz, Steffen Knoop, Ruediger Dillmann, and Raoul D Zollner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):322–332, 2007.
- [38] Bidan Huang, Miao Li, Ravin Luis De Souza, Joanna J Bryson, and Aude Billard. A modular approach to learning manipulation strategies from human demonstration. *Autonomous Robots*, 40(5):903–927, 2016.
- [39] Sang Hyounng Lee, Il Hong Suh, Sylvain Calinon, and Rolf Johansson. Autonomous framework for segmenting robot trajectories of manipulation task. *Autonomous Robots*, 38(2):107–141, 2015.
- [40] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5239–5246. IEEE, 2012.
- [41] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, DTIC Document, 1971.
- [42] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Safety*,

- Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1–6. IEEE, 2009.
- [43] Chris Burbridge and Richard Dearden. Learning the geometric meaning of symbolic abstractions for manipulation planning. In *Advances in Autonomous Robotics*, pages 220–231. Springer, 2012.
- [44] F. Wörgötter, E. E. Aksoy, N. Krüger, J. Piater, A. Ude, and M. Tamosiunaite. A simple ontology of manipulation actions based on hand-object relations. *IEEE Transactions on Autonomous Mental Development*, 5(2):117–134, 2013.
- [45] Florentin Worgotter, Eren Erdal Aksoy, Norbert Kruger, Justus Piater, Ales Ude, and Minijia Tamosiunaite. A simple ontology of manipulation actions based on hand-object relations. *Autonomous Mental Development, IEEE Transactions on*, 5(2):117–134, 2013.
- [46] Eren Erdal Aksoy, Minija Tamosiunaite, and Florentin Wörgötter. Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems*, 71:118 – 133, 2015.
- [47] Jeremie Papon, Alexey Abramov, Eren Erdal Aksoy, and Florentin Wörgötter. A modular system architecture for online parallel vision pipelines. In *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 361–368, jan. 2012.
- [48] M Schoeler, S Stein, J Papon, A Abramov, and F Wörgötter. Fast self-supervised on-line training for object recognition specifically for robotic applications. In *International Conference on Computer Vision Theory and Applications VISAPP*, January 2014.
- [49] T. Kulvicius, K. J. Ning, M. Tamosiunaite, and F. Wörgötter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Trans. Robot.*, 28(1):145–157, 2012.
- [50] Stefano Chiaverini and Lorenzo Sciavicco. The parallel approach to force/position control of robotic manipulators. *Robotics and Automation, IEEE Transactions on*, 9(4):361–373, 1993.
- [51] P. Soetens. RTT: Real-Time Toolkit. <http://www.orocos.org/rtt>, 2013.

-
- [52] Peter Soetens. *A Software Framework for Real-Time and Distributed Robot and Machine Control*. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium, May 2006. <http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf>.
- [53] Mohamad Javad Aein, Eren Erdal Aksoy, and Florentin Wörgötter. Library of actions: Implementing a generic robot execution framework by using manipulation action semantics. submitted.
- [54] E E. Aksoy, M. Tamosiunaite, R. Vuga, A. Ude, C. Geib, M. Steedman, and F. Wörgötter. Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots. In *IEEE International Conference on Development and Learning and Epigenetic Robotics ICDL-EPIROB*, pages 1–8, 08 2013.
- [55] Gennaro Chierchia. Syntactic bootstrapping and the acquisition of noun meanings: The mass-count issue. *Syntactic theory and first language acquisition: Crosslinguistic perspectives*, 1:301–318, 1994.
- [56] Steven Pinker. *Language learnability and language development, with new commentary by the author*, volume 7. Harvard University Press, 2009.
- [57] John C Trueswell, R Lila, et al. Learning to parse and its implications for language acquisition. 2009.
- [58] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [59] Andrei Haidu, Daniel Kohlsdorf, and Michael Beetz. Learning action failure models from interactive physics-based simulations. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5370–5375. IEEE, 2015.
- [60] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

-
- [61] J. Papon, A. Abramov, M. Schoeler, and F. Wörgötter. Voxel cloud connectivity segmentation - supervoxels for point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 2027 – 2034, 06 2013.
- [62] Johan Himberg, Kalle Korpiaho, Heikki Mannila, Johanna Tikanmaki, and Hannu TT Toivonen. Time series segmentation for context recognition in mobile devices. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 203–210. IEEE, 2001.
- [63] Maike Buchin, Anne Driemel, Marc van Kreveld, and Vera Sacristán. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 2011(3):33–63, 2011.
- [64] Amílcar Soares Júnior, Bruno Neiva Moreno, Valéria Cesário Times, Stan Matwin, and Lucídio dos Anjos Formiga Cabral. Grasp-uts: an algorithm for unsupervised trajectory segmentation. *International Journal of Geographical Information Science*, 29(1):46–68, 2015.
- [65] I. Markievicz, D. Vitkute-Adzgauskiene, and M. Tamosiunaite. Semi-supervised learning of action ontology from domain-specific corpora. In Tomas Skersys, Rimantas Butleris, and Rita Butkiene, editors, *Information and Software Technologies*, volume 403 of *Communications*, pages 173–185. Springer Berlin Heidelberg, 2013.
- [66] Ronald PA Petrick and Fahiem Bacchus. Pks: Knowledge-based planning with incomplete information and sensing.
- [67] A. Agostini, C. Torras, and F. Wörgötter. Efficient interactive decision-making framework for robotic applications. *Artificial Intelligence*, 2015. To Appear.
- [68] Sandor Szedmak, Enes Ugur, and Justus Piater. Knowledge propagation and relation learning for predicting action effects. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 623–629. IEEE, 2014.