March 2020

Functional Object-Oriented Network: A Knowledge Representation for Service Robotics

David Andrés Paulius Ramos
University of South Florida

Functional Object-Oriented Network: A Knowledge Representation for Service Robotics

by

David Andrés Paulius Ramos

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yu Sun, Ph.D.
Changhyun Kwon, Ph.D.
Xiaoning Qian, Ph.D.
Paul Rosen, Ph.D.
Sudeep Sarkar, Ph.D.
Yicheng Tu, Ph.D.

Date of Approval:
March 13, 2020

Keywords: Knowledge Representation, Domestic Robots, Bipartite Network, Graph Theory

Dedication

To my dear friends and family that have guided me along the way..

Thank you for being there for me.

This one's for you.

Acknowledgments

"Gratitude is not only the greatest of virtues, but the parent of all others"

—Marcus Tullius Cicero

I must first acknowledge and thank my family (spanning all across the globe – from Caribbean shores to large continents), especially my mother Alda, my father Wilson, and my brother Daniel, for their constant support, guidance and counsel which they continue to provide for me without end nor limit. You all have pushed me to keep learning and feeding my curiosity and to give it my best no matter what, while being mindful of all who have come before me. Daniel, I look forward to seeing you graduate from USF as a fellow alumnus and continue on to do great things.

Second, I must thank my dear friends, who have put up with me in my best and worst times, taught me and molded my core beliefs and values, and who have kept me company in life. From St. Kitts & Nevis, I have to mention the "guys": my best friends Annand (Ricardo), Jomo, TuWorld, and Darrione. I cherish all the times we hung out, talked about our problems, enjoyed drinks and many a "lime"; I look forward to seeing you continue to learn and grow. Thank you for your support, especially in times of my personal struggles. I cannot mention everyone, but I wish to thank and acknowledge the following friends of mine: Sheldon, Jagdish, Brascia, Shanequa, Jared, Orren, Saurabh, Jonelle, and Joash. From my time in St. Thomas, I have to "big up" my lunch crew (in alphabetical order): Ayana, Brandon, Cierra, Elvaneice, Hannah, Jamall, Jemanuel, Jenelle, Jiva, Latoya, Lorraine, Nicole, Sean, Tamara, and Yoelika. I must also acknowledge Daricia, Darnel, Phil, Jamila, Clyde, Odelmo, and Mandela. Thank you all for being there for such a major part of my life. From Antigua & Barbuda, I am grateful to Jeanine and her husband Darien; these words in my dissertation will never be enough to thank you for all that you've done for me and my family.

Here at USF, I have so many friends to thank. To my lab mates at the Robot Perception & Action Lab, I must first thank my seniors Yongqiang (a.k.a. Garfield) and John – as the rule of seniority lives on – and then to my other fellow peers: Troi, Ahmad, Mohammed, Tianze, Md Sirajus, and, more recently, Juan, Hailey, Maxat and Jean-Luc. Thank you for the many intellectual and

non-intellectual conversations that we kept in the lab. I sincerely treasure all the moments we have shared and all of the words of inspiration we have imparted to one another as we journey through the Ph.D. life. I also thank all of the undergraduate researchers who have worked with me: Roger, Vinh, Sarthak (Sid), Jennie, David (William), Kelvin, Sanjeeth and Joseph. It was a pleasure to be a mentor to each and every one of you. Next, I must thank fellow students that I've met along the way: Ilia, William, Parvaneh, Golam Bari, Love Kumar, and Odinaka. Finally, I would especially like to acknowledge someone special to me: Farhath (Farah). You have completely changed my life for the better; I can never thank you enough for your never-ceasing love and support.

Next, to the faculty and staff here at USF: I sincerely thank Dr. Yu Sun, my advisor and mentor, for taking a chance on me and for guiding me through my journey here. You have taught me about the qualities that a great professor should have and how to remain calm, even when deadlines loom and linger. I thank my committee members who have all played a role in the development of our research work: (in alphabetical order) Dr. Changhyun Kwon, Dr. Xiaoning Qian, Dr. Paul Rosen, Dr. Sudeep Sarkar, and Dr. Yicheng Tu. I am eternally grateful to the staff members in our department office: Gabriela Franco, Laura Owczarek, Jessica Pruitt, and Mayra Morfin. We appreciate you all and we continue to rely on you all to help us stay on track and to keep us sane. I would also like to thank former employees Yvette Blanchard and Kim Bluemer for their assistance when I started my program here. I want to thank José Ryan and Dr. Marbin Pazos-Revilla for being great mentors to me during my time in CSE Technical Support. I must especially thank Lashanda Lightbourne (a.k.a. Shanie), a former member of the CSE Department. Thank you so much for your constant words of encouragement whenever I came to visit you. Finally, thank you to all the professors who I have had the privilege of working with as their teaching assistant.

I want to acknowledge and thank other mentors and role models that I've had in my life: Ms. Eileen Grey, Mr. Dale Morton, Sir K. Dwight Venner (may he rest in peace), Dr. Wayne E. Archibald, Dr. Marc Boumedine, Dr. Maya Trotz, and Dr. Lynn Rosenthal (may he rest in peace).

At this moment, I want to thank everyone who I could not mention by name. By no means are your contributions invalid to my success nor do I disregard your support, and I take with me all of the experiences we've shared along the way. Many, many thanks to you all.

Lastly, by the grace of God, I have been able to finish this Ph.D. and continue forward to serve and help others. I look forward to experience what else is in store for me.

Table of Contents

List of Tables v

List of Figures vi

Abstract viii

Chapter 1: Introduction 1
1.1 Theory of Affordance 2
1.2 Related Works 3
1.2.1 Semantic Graphs 4
1.2.1.1 Activity Recognition and Inference with Graphs 4
1.2.1.2 Semantic Graphs for Sequencing of Skills or Events 5
1.2.1.3 Combining Semantic and Physical Maps 7
1.2.1.4 Context-free Grammars 8
1.2.2 Probabilistic Graphical Models 9
1.2.3 Cloud-based Distributive Representations and Systems 13
1.2.4 Cognitive Architectures 15
1.3 Contribution of Dissertation 17
1.4 Structure of Dissertation 18

Chapter 2: Functional Object-Oriented Network 20
2.1 Parts of a FOON 20
2.1.1 Network Data Structure 22
2.2 Accessing FOON 23
2.3 Creating a Universal FOON 24
2.3.1 Annotating FOON Graphs 24
2.3.2 Merging Subgraphs 26
2.4 Abstraction of FOON using Levels of Hierarchy 28
2.5 Future Work: Considerations for FOON Structure 32
2.5.1 Considering Multiple States of Objects 32
2.5.2 Recipe Look-up from FOON 32
2.5.3 Automatic Creation of FOON Graphs 33

Chapter 3: Network Analysis of FOON 34
3.1 FOON Statistics 34
3.2 Object Centrality 35
3.2.1 Measuring Centrality 36
3.2.2 Centrality Results 37

3.3 Motion Frequency 39
3.3.1 Motion Aliases 42

Chapter 4: A Motion Taxonomy for Manipulation Embedding 44
4.1 Motivation for the Motion Taxonomy 45
4.1.1 Conventional Representation of Motions 46
4.1.1.1 One-hot Encoding 46
4.1.1.2 Word Embedding 46
4.1.2 Grasp Taxonomies 47
4.2 Motion Taxonomy: Version 1 48
4.2.1 Motion Attributes 48
4.2.1.1 Contact Type 48
4.2.1.2 Engagement Type 49
4.2.1.3 Contact Duration 50
4.2.1.4 Trajectory Type 50
4.2.1.5 Manual Operation 51
4.2.2 Manipulation Codes: Version 1 51
4.3 Motion Taxonomy: Version 2 52
4.3.1 Describing Contact Type and Features 53
4.3.2 Describing Changes in Object Structure 55
4.3.3 Describing Trajectory of Motion 56
4.3.4 Translating Motions to Code 57
4.4 Evaluation of the Taxonomy 59
4.4.1 Experiment: Support for Motion Codes from Demonstration Data 59
4.4.2 Experiment: Observing Similar Motions from Demonstration Data 64
4.4.2.1 Methodology 64
4.4.2.2 Discussion 65
4.4.3 Experiment: Comparing Motion Codes to Word2Vec 67
4.4.3.1 Methodology 67
4.4.3.2 Discussion 68
4.5 Future Work and Ideas Yet Explored 71
4.5.1 Considering Other Features for Motion Codes 71
4.5.2 Using Motion Codes in FOON 72
4.5.3 Motion Generation from Motion Codes as Blueprint 72

Chapter 5: Task Planning through Knowledge Retrieval 73
5.1 Introduction to Task Tree Retrieval 73
5.1.1 Analysis on Task Tree Retrieval Algorithm 74
5.1.2 Example of Task Tree Retrieval 76
5.1.3 Novelty of Task Tree Retrieval 78
5.1.4 Modifying the Task Tree Retrieval 79
5.1.5 Motion Generation 79
5.2 Future Work and Ideas Yet Explored 79
5.2.1 Speeding Up Task Tree Retrieval 80
5.2.2 References to Source Recipe 80
5.2.3 Handling Unseen Cases of Items 81

Chapter 6: Generalizing Knowledge in FOON 82
6.1 The Concept of Object Similarity 83
6.2 FOON Expansion 83
6.2.1 Creating a FOON-EXP 83
6.3 FOON Compression 86
6.3.1 Creating a FOON-GEN 86
6.4 Evaluating the Usefulness of FOON Expansion and Compression 88
6.4.1 Methodology 88
6.4.2 Results for *FOON-65* 90
6.5 Limitations of *FOON-GEN* and *FOON-EXP* 91
6.6 Future Work and Ideas Yet Explored 92
6.6.1 Using Semantic Similarity in Real-World Scenarios 92
6.6.2 *FOON-EXP*: Handling Errors from Lexical Databases 92
6.6.3 *FOON-GEN*: Further Exploring FOON Compression 93

Chapter 7: Leveraging Robot's Capabilities with a Weighted FOON 94
7.1 Integrating Weights into a FOON 95
7.1.1 Deriving Weights for FOON 96
7.2 A Weighted Knowledge Retrieval 96
7.2.1 Finding the Optimal Tree 97
7.2.2 Analysis of the Path Tree Retrieval Algorithm 99
7.3 Human-Robot Collaboration 99
7.3.1 Human-assisted Manipulations 103
7.4 Experimental Results 105
7.4.1 Finding the Optimal Task Tree for NAO 105
7.4.2 Executing the Optimal Task Trees 106
7.5 Future Work and Ideas Yet Explored 108
7.5.1 Determining Weights for Robots 108
7.5.2 Improving the Path Tree Retrieval Algorithm 108
7.5.3 Robot-Robot Collaboration 109

Chapter 8: Concluding Remarks 110

References 113

Appendix A: HOW-TO: Using the FOON API 125
A.1 FOON_classes.py 125
A.2 FOON_graph_analyzer.py 128
A.2.1 Primary Functions in FOON_graph_analyzer.py 129
A.2.1.1 `_constructFOON` function 129
A.2.1.2 `_buildFunctionalUnitMap` function 129
A.2.1.3 `_createUniversalFOON` function 130
A.2.1.4 `_taskTreeRetrieval_greedy` function 130
A.2.1.5 `_calculateCentrality` function 130
A.2.1.6 `_expandNetwork` function 131
A.2.1.7 `_constructFOON_GEN` function 131

A.3 FOON_parser.py 132

Appendix B: HOW-TO: Using the FOON-view Tool 133

Appendix C: Copyright Permissions 134

About the Author End Page

List of Tables

Table 3.1	Statistics of current universal FOON, viz. *FOON-100*.	35
Table 3.2	Examples of motion aliases from various data sets.	43
Table 4.1	Manipulation Codes (Version 1)	52
Table 4.2	Manipulation Codes (Version 2)	58
Table 4.3	Mechanical characteristics described by the motion taxonomy.	60
Table 6.1	Statistics of expanded universal FOON (*FOON-EXP*) for *FOON-100* using WordNet.	86
Table 6.2	Statistics of expanded universal FOON (*FOON-EXP*) for *FOON-100* using Concept-Net.	87
Table 6.3	Statistics of *FOON-EXP* used in our experiments on *FOON-65* (threshold of 0.89).	90
Table 6.4	Average running times over all trials of random searches with an unexpanded network (REG), an expanded network using object similarity (EXP), and a generalized network with categories (GEN).	91
Table 6.5	Results of random-search experiment with an unexpanded network (REG), an expanded network using object similarity (EXP), and a generalized network (GEN).	91

List of Figures

Figure 2.1 A basic functional unit with three input nodes (in green) and two output nodes (in indigo) connected by an intermediary single motion node (in red) describing the action of stirring tea with sugar to sweeten it. 21

Figure 2.2 Illustration of a universal FOON combining knowledge from 100 instructional videos. 25

Figure 2.3 An illustration of a functional unit (Figure 2.3a) and its textual equivalent as a file (Figure 2.3b). 27

Figure 2.4 Illustration of two subgraphs which will be merged into a single FOON (best viewed in colour). 29

Figure 2.5 Illustration of merged FOON created from combining Figures 2.4a and 2.4b (best viewed in colour). 30

Figure 3.1 Illustration showing how one-mode projection works on an example subgraph (before as Figure 3.1a and after as Figure 3.1b). 38

Figure 3.2 Illustration of both degree centrality (Figure 3.2a) and Katz centrality (Figure 3.2b) for a level 1 one-mode projection of FOON. 40

Figure 3.3 Illustration of both degree centrality (Figure 3.3a) and Katz centrality (Figure 3.3b) for a level 3 one-mode projection of FOON. 41

Figure 3.4 Twenty (20) most frequent motions (85% of FOON motion nodes). 42

Figure 4.1 An illustration of the motion taxonomy (Version 1). 49

Figure 4.2 An illustration of the motion taxonomy (Version 2). 54

Figure 4.3 An illustration of the adapter used in the data collection in [1]. 61

Figure 4.4 Example of how PCA can be applied to recorded position data to derive prismatic bits of motion code for the 'stir' motion. 62

Figure 4.5 Example of how the axis-angle representation can be used to identify revolute bits of motion codes for the 'loosen screw' motion. 63

Figure 4.6 Matrix showing the Kullback-Leibler divergence values computed using DIM. 66

Figure 4.7 Graphs showing the 2D projection of vectors as a result of t-SNE from: a) motion codes with higher weight on contact features, b) motion codes with higher weight on trajectory features, c) motion codes with regular Hamming distance, and Word2Vec embeddings from d) Concept-Net, e) Google News, and f) Wikipedia 2018. 69

Figure 5.1 Using this subgraph describing the preparation of orange juice, a robot can derive the knowledge of cutting oranges (node in purple) using available objects in its surroundings (nodes in blue). 77

Figure 5.2 Task tree showing the steps needed to prepare halved oranges (highlighted in purple) using available objects (nodes in blue) using knowledge from Figure 5.1's subgraph. 78

Figure 6.1 An example of how expansion can fill in gaps of knowledge. 85

Figure 6.2 An example of how generalization through compression works on a subgraph. 89

Figure 7.1 Illustration of a weighted subgraph for the activity of making tea (where M is equal to 0). 101

Figure 7.2 Illustration of a weighted subgraph for the activity of making tea where (M is equal to 3). 102

Figure 7.3 An example of how task tree retrieval results can change depending on value of M. 104

Figure 7.4 Our experimental setup for demonstrating how a weighted FOON and HRC can be used with the NAO robot. 106

Figure 7.5 Graph showing the gradual improvement in success rates (y-axis) as M (x-axis) increases (best viewed in colour). 107

Abstract

In this dissertation, we discuss our work behind the development of the *functional object-oriented network* (abbreviated as FOON), a graphical knowledge representation for robotic manipulation and understanding of its own actions and (potentially) the intentions of humans in the household. Based on the theory of affordance, this representation captures manipulations and their effects on actions through the coupling of object and motion nodes as fundamental learning units known as functional units. The activities currently represented in FOON are cooking related, but this representation can be extended to other activities that involve manipulation of objects which result in observable changes of state. Typically, a FOON is created after annotating many demonstrations of how tasks are executed from start to finish and merging them all together to form a universal FOON. A robot programmed to use FOON will be equipped with the knowledge needed to solve manipulation problems, given a target goal as a node in FOON; we show how this procedure known as task tree retrieval can be executed by a robot. To circumvent possible physical limitations of the robot in executing manipulations (from the task tree retrieval procedure) successfully for cooking, we demonstrated how human-robot collaboration can also be used to overcome constraints. Complementary to the universal FOON creation procedure, we also investigated other means of learning concepts through semantic similarity as a solution to learning without the annotation of new demonstration videos. In addition to the retrieval algorithm, we also proposed motion embedding for representation of motions based on mechanical characteristics of said motions. Through this proposed representation, known as the *motion taxonomy*, we can solve the problem of ambiguity, which is inherent to human language when defining labels for motions or manipulations seen in demonstrations, by representing motions in a binary machine language.

Chapter 1: Introduction

Researchers over the past few decades have investigated how robots can be used to improve the quality of human life, primarily in the development of robots to perform tasks. The primary goal of automation using robots is to optimize daily human processes through which we can make tasks safer for humans (in high-risk domains or environments) or easier for those humans who cannot perform certain activities themselves (such as the elderly or disabled). In service robotics, we focus on robots that can aid or work alongside humans in human-centered environments, such as homes and offices. As a subset of service robotics, domestic robots are designed, built and programmed to interact with humans and to assist them in activities of daily living (ADL) such as cooking and cleaning. However, more recently, roboticists pay particular attention to developing robots that act intelligently rather than programming them with fixed functionality or primitives; rather, as intelligent agents, much like in AI, these robots can understand their given tasks, identify who or what are in its environment (which may act as constraints upon its manipulations), and to determine a plan of action to tackle an existing problem. While executing its manipulations, the robot should have the ability to understand the effects or consequences of its actions upon others and the world. In developing these robots, researchers build systems that combine knowledge representation, reasoning, and retrieval which are based upon on several psychological concepts and theories based on neuroscience and human cognition.

In this dissertation, I present approaches we have developed and proposed for creating effective representations of knowledge that can be used by domestic robots to perform their tasks intelligently and safely, which robots can use to understand its own intentions and those of humans that are around it. First, we will review the *functional object-oriented network* (FOON) – a knowledge rep-

This chapter was partially published in [2] and [3]. Permission is included in Appendix B.

resentation that takes the form of a graph data structure which captures the relationship between objects and motions to produce effects through manipulations.

Definition 1. *As per the definition in [3], a knowledge representation can be defined as "a means of representing knowledge about a robot's actions and environment, as well as relating the semantics of these concepts to its own internal components, for problem solving through reasoning and inference."*

This knowledge representation is driven by prior work by senior colleagues [4, 5, 6] and originally inspired by the theory of affordance [7]. A FOON is constructed by annotating demonstrations of manipulations in ADL. For the time being, we have focused on the task of cooking, where a robot will have the knowledge of what utensils, ingredients or objects can be used together to create or prepare recipes. Inspired by our experiences with FOON, we will also review motion embedding using the *motion taxonomy*, which we proposed for representing motions in an attribute-level space for motion classification, recognition, annotation, and possibly generation. This taxonomical embedding of motions can be used for defining meaningful motion labels that can be used in classification algorithms and in representations like FOON.

1.1 Theory of Affordance

FOON, especially the intuition behind its graphical structure, is inspired by the *theory of affordance*, which was originally proposed by psychologist James J. Gibson in 1977 [7]; this theory states how objects innately have properties through which certain actions are *afforded* to the users (or manipulators), meaning that these properties define or describe how objects can be manipulated or used. For instance, in the household, objects such as knives have sharp edges, affording us the action of cutting; they also have handles that *afford* us the action of grasping or holding, which is a property also shared in other objects such as mugs, pots and pans.

The theory of affordance is supported by several studies in neuroscience and cognitive science, which have all demonstrated that the mirror neurons in human brains congregate visual and motor responses [8, 9, 10]. More specifically, mirror neurons in the F5 sector of the macaque ventral premotor cortex fire during both observation of interacting with an object and action execution,

but they do not discharge in response to simply observing an object [11, 12]. More recently, Yoon et al. [13] recently studied affordances associated to pairs of objects positioned for action and found an interesting so-called "paired object affordance effect", where the response time by right-handed participants was faster if the two objects were used together, where the active (manipulated) object was to the right of the other. Further studies by Borghi et al. [14] also corroborate the functional relationship between paired objects and compared it with the spatial relationship and found that both the position and functional context are important and related to the motion; however, the motor action response was faster and more accurate with the functional context than with the spatial context. A comprehensive review of models of affordances and canonical mirror neuron system can be found in [15].

In summary, the findings from these studies indicate that there are strong connections between the observation of objects and the functional motions. Further, functional relationships between objects are directly associated with the motor actions. This interesting phenomenon can be observed in human daily life; when humans are performing tasks, they pay attention not only to objects and their states, but also to object interactions caused by manipulation. The manipulation reflecting the motor response is tightly associated with both the manipulated object and the interacted object. It has even been shown that affordance can be used to infer the type of action or objects that are being used. Helbig et al. [16] showed how an object that is occluded from view can be inferred solely based on the type of grip or grasp that is observed.

1.2 Related Works

We now discuss research works that developed knowledge representations or models for robot learning and manipulation. Several of these works are also based on the theory of affordance. A good overview of knowledge representations can be found in [3]; the following subsections are excerpts from this paper.

1.2.1 Semantic Graphs

Graphs are very popular for representing information quite simply because we can display knowledge in a graphical form that can be interpreted and verified visually by humans. Probabilistic models can also be represented graphically and make excellent inference tools. With probabilistic graphical models, edges would describe the likelihoods of certain variables as nodes causing others to occur. However, in this section, we will be referring to another subset of graphs referred to as *semantic graphs*, whose nodes and edges describe semantic concepts and details between entities as observed in demonstrations. Spatial concepts, for instance, can be described by semantic graphs, where nodes can describe objects within a scene, and edges describe commonality or contextual relationships between objects in terms of position (one object may hold another object, one object may be on top of another object, et cetera). Some graphs also embody temporal relations, where two or more particular events are related by time (e.g. one event must occur before another). Basically, these networks can be used for compressing details and capturing relationships as needed by a robot.

1.2.1.1 Activity Recognition and Inference with Graphs

One of the major problems in robot learning has been in learning to recognize activities to facilitate the transfer of knowledge to robotic systems. A major component of activity recognition and understanding is predicting an ongoing activity or action as seen in a video demonstration. Knowledge extraction is mainly done through the processing of activity-based videos and images or through interpreting sensor data from demonstrations either done by the robot or human demonstrators. Techniques such as optical flow can also be used for identifying motion patterns to then recognize motion types or primitives. These elements can be used as context clues for inference. Previous work focused on understanding the activity taking place with the use of the primary portion of such videos to recognize the likely activity and results which would be implied by it [17, 18, 19], especially for predicting the next action which would take place in a sequence of actions [20]. Semantic graphs have been used for representing affordances based on how objects are used with one another based on visual cues and spatio-temporal relatedness.

Segmentation techniques can be applied to images to identify the objects being used in demonstrations. These segmented "patches" can be used for labelling nodes in semantic graphs. For example, Aksoy et al. [21, 22] created these semantic graphs after segmentation. Their focus was in understanding the relationship between objects and hands in the environment and generalizing graphs for representing activities and identifying future instances of these events. This approach can be classified as unsupervised learning since there is no explicit indication of what the objects are; objects instead are solely encoded based on manipulations in matrices, which they refer to as *semantic event chains* (SEC). These structures capture the transitions in segment relations (temporal information), which are then generalized by removing any redundancies in activities, to be used in recognizing similar events. They characterized spatial relations of objects as non-touching, overlapping, touching, or absent within each matrix entry and as edges which connect image segments. Sridhar et al. [23] also used segmentation to separate the scene into "blobs" (similar to the patches in [21, 22] and cluster them as a semantic graph, based on the objects' usage in videos, for affordance detection. Their semantic graphs are called *activity graphs*, structures which describe the spatial (whether objects are disconnected, found in the surrounding area, or touching) and temporal (relativity with respect to episodic events) relationships in a single video. With such graphs, similarity between activities can be measured even with varying object instances, orientations, hand positions, and trajectories. Zhu et al. [24] focused on segmenting the tool and the object it is being used on to create a spatial-temporal *parse graph*. Within these graphs, they capture the pose taken by a demonstrator, the observed grasping point of the tool, the functional area of the tool that affords an action, the trajectory of motion, and the physical properties (such as force, pressure or speed) that govern the action. These graphs can then be used to infer how objects can be used based on prior demonstrations.

1.2.1.2 Semantic Graphs for Sequencing of Skills or Events

Semantic graphs may also been used for task execution in the form of skills, containing knowledge that can be used by robots for manipulations. In these structures, nodes represent objects and action types. Several researchers have taken approaches to learning object affordance and

representing them in this manner. For example, Ramirez-Amaro et al. [25, 26, 27] used learning by demonstration to teach robots about manipulations obtained directly from demonstrations, and they describe it as a transfer of skills from the demonstrator to the robot; upon observation of a demonstration of a skill, the robot then imitates the action performed by a human demonstrator. This sense of “transfer learning" however is different to the traditional sense within the machine learning community [28]. They can create semantic graphs as trees with knowledge in the form of transferable skills needed to execute three challenging kitchen tasks. This knowledge is directly extracted from human demonstrators and it allows the robot to perform the exact methods needed to imitate the demonstrator in manipulating objects. Human activities are learned based on several attributes: 1) the motion made by the hand, 2) the object(s) being moved and manipulated by the hand, and 3) the object(s) which these held items are being used and acted on, and they are presented as ordered pairs to train their inference engine. Once obtained, these semantic rules, grounded in Prolog, can be used in reasoning and future understanding of demonstrations through inference; these properties were applied to a decision tree classifier to automatically gain knowledge and rules from new demonstrations.

FOON is particularly inspired and akin to a representation known as Petri Networks (or simply Petri Nets) [29]. Petri Nets were originally intended for illustrating chemical reactions, and they have been shown to be applicable to other domains such as robotics and assembly. Petri Nets are networks with two types of nodes: *place* nodes and *transition* nodes. Place nodes represent states of objects or entities, and transition nodes represent events or actions which cause a change in state. The term for state change with respect to Petri Nets is *firing* of transitions. Typically, all place nodes must be present for transitions to fire, therefore enforcing an implicit ordering of actions and behaviours. Costelha et al. [30, 31] used Petri Nets for representing robot tasks over other methods such as Finite State Automata (FSA) which require more memory and a larger space of representation and its limitation to single-robot systems. Petri Nets, on the other hand, can represent concurrent system actions and sharing of resources. The implicit ordering of events allows them to filter out specific plans which can never happen or those which should be avoided.

They created *Petri Net Plans* (PNP), which are essentially a combination of ordinary actions and sensing actions using control operators.

Similar to context-free grammars are *object-action complexes* (OAC, pronounced like “oak") [32, 33, 34, 35]. This representation’s purpose is to capture changes in state of the environment in a formal structure which can be used for task execution. OACs combine high-level planning and representation of the world with low-level robot control mechanisms called instantiated state transition fragment (ISTF). An ISTF can be seen as minute, lower-level constructs, which can be put together like context-free grammars, for a concrete understanding of an action’s effects before and after a motor program (i.e. action) is executed; OACs can be created after learning a variety of ISTFs. ISTFs are generalized to only contain the object-state changes which are relevant to an action tuple (identified through methods described in [36]), as ISTFs can contain object-states which may or may not be affected or caused by a specific action. Given a set of object affordances and relations learned, an associative network can be used for encoding and retrieving the state change that will occur from a certain OAC permutation.

Other approaches so far have attempted to map high-level manipulation skills to graphical structures. Instead of focusing on manipulations, Konidaris et al. [37] chose a different representation for trajectories as skills. These researchers introduced an algorithm for learning skills from demonstrations, focusing primarily on motion trajectories from tasks, called CST (for *Constructing Skill Trees*). Motion trajectories can be broken down using change-point detection, and these smaller trajectory components are referred to as skills. The aim of change-point detection is to find the point(s) at which there is a significant or observable change in trajectory. After successfully segmenting the trajectories into skills, these learned skills can be combined together as *skill trees* for potentially novel manipulations by appending skills into one executable skill.

1.2.1.3 Combining Semantic and Physical Maps

Semantic graphs can also take the form of *semantic maps*, which are special graphs that relate spatial or geometrical details (such as morphology of space, position and orientation of objects, geometry of objects as models, and any positions of special places of interest) to semantic de-

scriptions (such as the purpose of objects). These spatial features can be acquired from SLAM modules, including properties such as sensor readings or features, orientation, and absolute or relative positioning of objects or landmarks; through SLAM, the robot can obtain a map of the environment that uses the contextual information to particularly highlight instances of different objects or points of interest that lie there and to identify where they are, for instance. Semantic maps have also been used in identifying grasps by using geometrical features about the objects [38, 39]. An example of how semantic maps can be created was proposed by Galindo et al. [40], which integrates causal knowledge (how actions affect the state of the robotâĂŹs environment) and world knowledge (what the robot knows about objects around, their properties, and their relations), using two levels of information: the spatial box (S-Box) and terminological box (T-Box); they mark the physical location of objects at the sensor level as well as note the free space in rooms with S-Box, while the innate properties of these objects are linked using ontologies with T-Box. *Semantic object maps* (SOM) [41, 42] also serve a similar purpose to combine geometric data with semantic data to answer queries to determine whether a certain action can be executed given present circumstances in its environment. For example, a Room instance can be inferred to be a kitchen if there are items within the environment that are typical of a kitchen, such as a stove or a fridge. With regards to creating semantic maps through human interaction, works such as [43, 44] and [45] aimed to develop HRI systems to impart knowledge to a robot about its surroundings: what lies around it and the conceptual knowledge tied to these elements. Both systems use audio for interacting with robots; in addition to this speech recognition system, [45] combined a tangible user interface and a vision system with a robot's modules for motion planning and mapping to compile and create a knowledge base which a robot can then use for its navigation through its environment.

1.2.1.4 Context-free Grammars

Context-free grammars are also an effective way of constructing or representing semantic graphs or structures, as they guarantee completeness, efficiency and correctness. A context-free grammar defines rules to creating semantic structures and sub-structures as strings using its own

symbols (called *terminals*) defined within a finite set called an *alphabet*. These terminal symbols can be used when substituting variable symbols called *non-terminals*; the substitution process is described by production rules, which allow us to generate sentences. With such a formal definition of a context-free grammar, researchers have been able to define rules that describe concepts such as manipulation/action types which can then be useful for defining plans that robots can use for execution and also for the composition of skills into sub-skills. One such example of a context-free grammar was proposed by Yang et al. [46, 47, 48, 49], who studied how manipulations in activities can be represented through grammar in the form of combinatory categorial grammar (CCG) and then broken down into visual semantic graphs. This grammar vividly describes a specific action, the items being used, as well as the consequence of performing such an action [46], and each action can also be effectively broken down into smaller sub-actions or sub-activities. These parse trees are built from demonstrations can then be used to form manipulation action tree banks. The high-level representation serves as a symbolic representation of the manipulations which describe each step required to solve a given problem. Using said context-free grammars, they also developed *manipulation action tree banks* [48] to represent action sequences as tree data structures that can be executed by a robot. Equipped with action tree banks, a robot can use the knowledge gathered from multiple demonstrations to determine the actions it needs to take, in the form of a tree, for a given manipulation problem.

In a different approach that also uses context-free grammars, Dantam et al. [50, 51] also formulated robot primitives and control policies using their own representation called *Motion Grammars* (MG). A parse tree can be constructed to reflect the procedures being executed and they can be broken down by a motion parser to create sub-tasks until they have been satisfied, similar to the representation introduced by Yang et al.

1.2.2 Probabilistic Graphical Models

In this section, we focus on learning approaches that use *probabilistic models* as their base of knowledge, which assume that a robot's world and the actions it can possibly execute are not discrete but indeterminate by nature. In other words, the robot's world is governed by likelihoods

and uncertainty, and these likelihoods are captured as probabilities grounded in such models. These models therefore can be used for representing knowledge needed by robots when it comes to recognizing activities through a basal understanding of the effects of its own actions on its environment. Although these models are examples of machine learning algorithms, they are fit to learn high-level concepts and rules for inference; other machine learning techniques that do not focus on these high-level rules would be considered as implicit representations of those rules.

Bayesian Networks (BN) in research studies are mainly used for capturing the effects of actions upon objects in a robot's environment. When capturing such effects, a robot would be presented with demonstrations of observable action and effect pairs in order to learn the relationships between them. These relationships can be taught through learning by demonstration, and it can be classified into two subcategories: *trajectory-level* learning and *symbolic-level* learning [52]. Trajectory-level learning is a low-level representation approach which aims to generalize motions on the level of trajectories and to encode motions in joint, torque or task space, while symbolic-level learning looks at a high-level representation which focuses on the meaning behind skills in activity learning. The robot's interaction with its environment serves to either learn new motor primitives or skills (trajectory-level) or to learn new properties associated with the type of grasp they make or the skills they use, the object's physical features, and the effects that occur from executing an action (symbolic-level). In works such as [53, 54] [55, 56, 57] [58], a robot can use basic, pre-programmed motor skills (viz. grasping, tapping or touching) to learn about relationships between an object's features (such as shapes, sizes or textures) and features of its actions (such as velocities and point-of-contact). The controllers of these skills are tuned by the robot's experiences and exploration with its environment, and the causality of these actions and their effects upon the world, based on object features, can be represented through a BN. The robot can use the knowledge it has gathered from interacting with objects and performing fine-tuning to select the appropriate action that achieves a human demonstrator's result. Similarly, Jain et al. [59] and Stoytchev et al. [60, 61] used these networks to learn about the effects of actions on objects based on demonstrations with tools. Their BNs were built based on geometric features relevant to a tool's function (and tools similar to it), which they coined as functional features, for predicting the effects of tools unknown

to the robot with the learned model. For instance, objects used for cutting have a sharp edge as a functional feature, and those used as containers have a non-convex shape for holding matter or substances; once the robot can identify these features, it can use the trained model to predict the results of specific actions. The tools' effects are given by the target object's displacement, the initial position of the tool relative to the target object, and the target velocity after impact was made on the tool's functional feature. A BN can also be used with other modalities of data such as speech input for grounding actions to their effects [62]. Instead of learning object-action effects, BNs can also describe the likelihoods of object-object interaction between specific pairs of objects as learned from observing human behaviour [5]. These affordance models are particularly useful in improving both activity recognition and object classification and teaching robots to perform tasks using paired objects.

With regard to Markov Networks (MN), instead of a cause-effect relationship as inherently represented in Bayesian Networks, researchers can focus on learning dependencies between concepts useful for learning from demonstrations and identifying future cases of actions or activities, which can particularly be useful for a robot in learning new concepts or safely coordinating with others working in the scene. As an example of activity recognition with MNs, Kjellström et al. [63, 64] used CRFs to perform both object classification and action recognition for hand manipulations. Their reasoning behind the simultaneous classification of both objects and manipulations comes from: 1) the sequence of an object's viewpoints and occlusion from the hand indicate the type of action taking place, and 2) the object's features suggest the way the hand will be shaped to grasp it. They use factorial conditional random fields (FCRF) [65] to map this two-way relationship between object features and possible action types. FCRFs have the advantage of mapping the relationship between the data level (features found in observations) and the label level (object types and properties and their relatedness with actions), thus effectively capturing affordances suggested by the hands and objects. A similar approach is taken in [66] to identify activities based on objects and their proximity to hands in a scene using CRFs. Prior to this work, in [67], this association was described using text descriptions, which they denote as functional object string descriptors, which significantly perform better than using purely appearance-based descriptors

for recognizing similar events or activities. Using CRFs to represent the spatio-temporal relationship between objects, denoted by functional classes, improved over their previous approach of performing activity recognition with string kernels.

Another example of spatio-temporal representation of activities to objects was done by Koppula et al. [68], who used MRFs to describe the relationships present in the scene between activities and objects. By segmenting the video to the point of obtaining sub-activity events, they can extract a MRF with nodes representing objects and the sub-activities they are observed in and edges representing: 1) affordance-sub-activity relations (i.e. where the object's affordance depends on the sub-activity it is involved in), 2) affordance-affordance relation (i.e. where one can infer the affordance(s) of a single object based on the affordances of objects around them), 3) sub-activity change over time (i.e. the flow of sub-activities which make up a single activity), and 4) affordance change over time (i.e. object affordances can change in time depending on the sub-activities they are involved in). They proposed using this model for the purpose of recognizing full-body activities occurring in videos collected for their Cornell Activities Dataset[1] (CAD). Following [68], they investigated how they can anticipate or predict human behaviour using CRFs to ensure that a robot reacts safely in [69]. A special CRF, called the anticipatory temporal CRF (ATCRF), can be built after identifying object affordance for a particular action type and can effectively describe all possible trajectories of human motion and sub-activities that are likely to be taken as time goes by.

Using first-order logic statements are effective for reasoning and inference; taking advantage of such logical expressions, a MLN effectively represents a systematic encoding of the robot's world. A MLN can be thought of as a knowledge base, as these logical statements can be used for reasoning and drawing conclusions based on what a robot sees and observes. For instance, with regards to activity recognition using affordance, Zhu et al. [70] used a knowledge base, in the form of a Markov Logic Network, for inferring object affordances in activities which are suggested by the pose of the human demonstrator in videos. They can do the inverse by predicting the objects and actions occurring in a given scene based on the pose of the human demonstrators with relatively great performance. This could only be done after they collected a large amount of information

[1]Cornell Activities Dataset – http://pr.cs.cornell.edu/humanactivities/

about these usable objects and affordances as features, but there is no need for training multiple classifiers for each object-based task to identify each type of detectable activities as typically done with other machine learning approaches. A MLN such as theirs can be used alongside other components for activity recognition to predict human intention and to enforce safe behaviour within a human-robot collaborative environment. KnowLang [71, 72], proposed by Vassev et al., is a knowledge representation that was developed for cognitive robots where the power of AI's logical expressions of the world with what they actually perceive in their world. It also combines first-order logic with probabilistic methods which they can use for defining explicit knowledge for the robot. However, when making certain decisions in which lies uncertainty, statistical reasoning through the use of Bayesian Networks makes the process more reliable through reasoning on beliefs. Experiences can be reflected through probabilities, and such distributions are likely to change based on what the robot sees or acts.

1.2.3 Cloud-based Distributive Representations and Systems

When it comes to comprehensive knowledge representations that are based over the cloud, prominent examples include RoboEarth and RoboBrain. RoboEarth [73, 74], referred to as the "World Wide Web for robots", is an ongoing collaborative project aiming to create a cloud-based database for robots to access knowledge needed to solve a task. RoboEarth was first proposed as a proof-of-concept to show that cloud robotics would greatly simplify robot learning. RoboEarth provides an ontology for storage of semantic concepts and a method for accessing knowledge through a software as a service (SaaS) interface, where computational inferences and mappings can be done remotely. As a collaborative system, RoboEarth allows robots to archive its own experiences (such as object types observed, motion plans successfully or unsuccessfully used, robot architectures, etc.) for recall and reuse by other capable robots. This database would contain massive amounts of data (in the form of object models, SLAM maps [75], semantic maps, etc.) which can be used for tasks such as object recognition, instance segmentation and path-planning. Related to RoboEarth is another promising project headed by Rapyuta Robotics[2], a company which

[2]Rapyuta Robotics – https://www.rapyuta-robotics.com/

now deals with cloud robotics solutions. Rapyuta Robotic's system called *Rapyuta*, named after the movie from Japan's Studio Ghibli, was first introduced in [76] and then in [77] as a platform as a service (PaaS) interface for robots. It acts as an open-source middleware for accessing resources from the web such as the aforementioned RoboEarth repository and ROS (Robot Operating System) packages. Additionally, it reduces the processing done by the robot by offloading computations to the Cloud. Robots can also communicate and share information with one another through this PaaS system. This project has since evolved into the development of their cloud robotics platform for corporate solutions.

Results from RoboEarth led into the development of openEASE[3], also by Beetz et al. [78] (EASE being an abbreviation for *Everyday Activity Science and Engineering*). builds upon RoboEarth as a web-based interface and processing service that equips robots with knowledge from prior experiences (similar to accessing memory) and reasoning capabilities in the form of semantically labelled activity data. A robot using openEASE will have access to: 1) knowledge about a robot's hardware, its capabilities, its environment and objects it can manipulate, 2) memorized experiences which a robot can use for reasoning (why it did an action, how it did it, and what effects the action caused), 3) annotated knowledge obtained from human demonstrations. Queries and statements are formulated using ProLog, which can be sent through a web-based graphical interface or through a web API usable by robots; they allow robots to acquire semantic information and meaning to sensor input and to data structures used for control purposes. As a component to this project, Tenorth et. al [79, 80, 81] presented KnowRob as a knowledge processing system for querying the openEASE knowledge base using Prolog predicates. KnowRob combines various sources of knowledge such as web pages (methods from instructional websites, images of usable objects, etc.), natural language tasks, and human observations. A robot can ground the knowledge from KnowRob to a robot's perception/action system and its internal data structures through a symbolic layer referred to as "virtual knowledge bases". Through ProbCog [82], a statistical relational learning and reasoning system, models such as Bayesian Logic Networks [59] or Markov Logic Networks can be built for representing the state of the robot's current context. KnowRob is

[3]openEASE – http://www.open-ease.org/

built within another tool known as CRAM (short for *Cognitive Robot Abstract Machine*) [83, 84], a software toolbox for the design, implementation and deployment of robots using its own CRAM Plan Language (CPL). CPL is inspired by Common Lisp and Prolog for the expressive specification of concurrent, sensor-guided reactive behaviour, or in simpler terms, how a robot should react to certain sensory events or changes in belief state.

Another noteworthy technology that deals with knowledge gathering and sharing is RoboBrain[4]; Saxena et al. [85] introduced RoboBrain in 2014 as a means of massively collecting concepts which are learned from automatic gathering of data from the Internet, simulations, and robot experiments. This differs to the RoboEarth/openEASE representation in the fact that RoboBrain uses graphs for encoding knowledge, while RoboEarth and its internal components use propositional logic and statements for defining concepts and relations in a working space. The information is represented as a graph, where nodes represent concepts (such as images, text, videos, haptics data, affordances, deeply-learned features, etc.) and edges represent the relationships between such concepts. RoboBrain connects knowledge from popular sources such as WordNet [86], Wikipedia, Freebase, and ImageNet [87]. They manage errors in knowledge collection using crowd-sourcing feedback as well as beliefs that reflect the trust given to certain knowledge sources and the correctness of concepts and their relations. To retrieve knowledge, the Robot Query Language (RQL) can be used for obtaining a subgraph describing the activity of executing a certain task. Unlike the case with openEASE, it was not demonstrated how a robot can execute the method reflected by a subgraph; however, the knowledge gathered nevertheless can be quite useful for task planning, instance identification, and inference.

1.2.4 Cognitive Architectures

Another popular representation that has drawn influences from psychology and neuroscience is cognitive architectures. A *cognitive architecture* is a model that attempts to explain the psychological theories behind human cognition and the processes, mechanisms and modules that are pivotal to cognition and behaviour. Many architectures have been proposed and extensively studied based

[4]RoboBrain – http://robobrain.me

on different cognitive theories, including the likes of ACT-R [88], Soar [89, 90], CLARION [91], and EPIC [92]. Every architecture differs from one another in how knowledge is acquired, represented, retained (through long- and short-term memories), and transmitted within its internal cognitive structures since they all follow their own respective theories of human cognition; despite their differences, however, they all aim to answer the question of how human cognition works. Much like knowledge representations for robots, cognitive architectures modularize different components that are central to thoughts, perception, inference, and action and have them connected with one another based on cognitive theories.

Typically, cognitive architectures emphasize on how knowledge is retained and used as either long- or short-term memory, where long-term memory can refer to concepts (goals or descriptions of an entity's environment), while short-term memory refers to instances of such concepts. Each of these concepts and skills are learned, retained, and activated once their arguments have been fulfilled by identifying them through the robot's perception system. Architectures such as Soar [93] and ICARUS [94, 95, 96] have been used to illustrate how skills and concepts are learned from navigating throughout its environment or through human interaction. Within robotics, cognitive architectures are extensively studied in the field of developmental robotics as a means of understanding how we develop sensory and motor skills as an infant. Such studies look at how each internal component is connected with one another so that a robot can develop new skills and acquire knowledge about its problem domain. They can learn concepts such as object instance identification and location. These skills (i.e. its abilities to manipulate objects) are learned as long-term components which can be used in conjunction with other skills for performing tasks on the robot's environment. Through a developmental approach, works such as [97, 98, 99, 100] investigate how skills are developed through human-robot interaction (HRI). In such studies, a human assistant would interact with robots to teach them about the relationships between its actions, effects, and its own internal representation of the world. For example, in Ivaldi et al. [99], a robot was taught about object affordance through the assistance of a caregiver who supervises the robot's actions much like a parent would. The caregiver can give the robot different commands such as looking, grasping, pushing or more complex actions like placing objects on top of another.

Through HRI, the robot learns how to identify novel objects by showing the robot what the items of focus are without any prior knowledge of what they look like. Once the robot has the knowledge of those objects, the robot can proceed to learn about actions and the action's effects while gaining positive or negative feedback to indicate whether the robot performed the task correctly or not. In summary, cognitive architectures not only aim to equip robots with the necessary knowledge to perform its duties, but they ultimately aim to explore how our cognition as humans work. Retaining a memory of experiences is crucial to learn semantic and symbolic concepts and to reason for solving problems.

1.3 Contribution of Dissertation

This research work contributes the following to the robotics community:

1. We propose and define a simple yet effective knowledge representation for robots that use the concept of affordance as a driver to problem solving.

2. We provide a data set of 100 demonstration videos from several sources of various cooking activities, which can be used for video understanding problems. In related work, we have presented a pipeline for video understanding using FOON as reference. This data set is publicly available and accessible through our project's website and through our FOON API repository (which is described in detail in Appendix A).

3. We propose a framework for using FOON with real robotic systems, which aims to leverage the performance of robots based on their physical capabilities. We show how FOON can be used in multiple entity task planning and execution with the case of a human assistant.

4. We propose a motion taxonomy – a representation of motions for use in motion recognition and generation tasks that deviates from human language – that can be used to generate motion embeddings in the form of motion codes. With a more adequate representation of motions through motion codes, researchers are better able to define classes and features that explicitly define and distinguish motions from one another.

1.4 Structure of Dissertation

This dissertation is structured as follows:

- In Chapter 2, we introduce the FOON representation. We define its basic structure, how it is constructed, and how it can be accessed for use.
- In Chapter 3, we delve into graph analysis on FOON – specifically network centrality to determine important nodes needed for cooking procedures and activities. We also review other statistics of FOON such as the number of nodes within the network. These values correspond to our latest version of FOON comprising of 100 demonstration videos, which we refer to as *FOON-100* within this dissertation.
- In Chapter 4, we introduce the *manipulation motion taxonomy* to encode motions into a binary string label (that can then be broken into a vector) and to embed motions in an attribute space that describes the motion's characteristics. We show how the natural clustering of motions as motion codes support real data and how these embeddings are better suited than Word2Vec [101], which is a popular approach to learning word embedding from natural language.
- In Chapter 5, we discuss the *task tree retrieval algorithm*, which is an algorithm that a robot can use in task planning to derive the steps it needs to execute to produce a target object, given knowledge about its surroundings.
- In Chapter 6, as an alternative to annotating new videos manually (or, at best currently, semi-automatically [102]), we introduce two techniques to expand the knowledge in FOON through two methods: FOON *expansion* and FOON *compression* (or generalization). We present up-to-date results of these techniques with our significantly larger *FOON-100*.
- In Chapter 7, as a solution to robotic programming with FOON, we present human-robot collaborative task planning and execution with FOON. Due to the difficulty in programming a robot to perfectly execute the necessary manipulations outlined in FOON, we propose human-robot collaboration as a method to circumvent the limitations imposed by the robot's structure and architecture.

- In Chapter 8, we end our discussion with concluding remarks on the state of knowledge representations for robots and what it means for FOON.

Chapter 2: Functional Object-Oriented Network

The *functional object-oriented network* (FOON) represents manipulations as seen in cooking activities (and with possible extension to other manipulation-centric tasks or domains) by capturing the objects and the activity's motions within a graphical structure. Originally proposed in [2], we formally introduced FOON as a graphical knowledge representation that represents high-level concepts related to human manipulations for service robotics tasks. This representation is motivated by the theory of affordance [7], wherein it describes the underlying uses and effects of objects afforded to the robot, which are innately depicted in FOON though edges connecting objects to actions. As we have introduced before, the purpose of this knowledge representation is to serve as a source of knowledge for a robot to determine how it can go about solving a problem. In this chapter, we talk about the basics of FOON: the types of nodes used, how they represent manipulations, how we can construct a FOON from demonstration, and how FOON can be accessed by interested individuals.

2.1 Parts of a FOON

As given in the definition before, a FOON is primarily based on the graph data type, which is comprised of nodes (or vertices) and edges. To adequately represent activities, a FOON contains two types of nodes: object nodes and motion nodes. Object nodes symbolize any object that is manipulated actively or passively within the activities in FOON, while motion nodes symbolize the type of manipulation that connected object nodes are participating in at a given period of time. An object node is defined by an *object type* (describing what object it is), a *state type* (describing the state or condition in which the object is observed to be in), and its *contained items/ingredients*

This chapter was published in [2] and [103]. Permission is included in Appendix B.

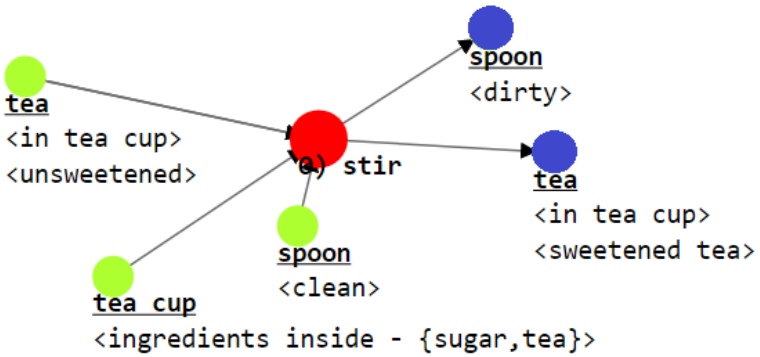


Figure 2.1. A basic functional unit with three input nodes (in green) and two output nodes (in indigo) connected by an intermediary single motion node (in red) describing the action of stirring tea with sugar to sweeten it.

(describing what objects form as components to another object acting as a container or medium). Prior to this work, object nodes were only assigned a single state type, which reflected the most important or influential state at a given point in time; however, we have now made revisions such that objects can be defined with multiple states. Multiple states would capture a mixture of physical properties and the location of the object as it is found or observed in the environment. A motion node is defined by a *motion type* (an identifier describing what type of action it is); this motion type can either be a manipulation (e.g. stirring, cutting, or pouring) or non-manipulation action (e.g. baking, cooking, frying – where cooking utensils are immobile). These motion nodes not only reflect cooking actions, but they can also extend to manipulations in other domains, such as assembly or manufacturing processes. However, there is one point to note concerning the prevalence of nodes in a FOON: object nodes are kept unique based on their labels, while motion nodes are not and thus can be duplicated. The reason for duplicate motion nodes is that there may be different ways of using objects with the same motion type, so it is necessary to have multiple instances of motion nodes to capture the variations of actions that could happen in cooking.

As an example shown in Figure 2.1, which illustrates the task of stirring a cup of tea using a spoon as a functional unit, the active object in this case would be a *spoon* object that acts upon a passive object *tea cup* which contains the ingredients *tea* and *sugar*. The stirring manipulation is represented here with a motion node with the label *stir*. As a result of this action, the *tea* changes

state from *unsweetened* to *sweetened*. The joint representation of both object and motion nodes make FOON a bipartite network. As with typical bipartite networks, where an edge connects two nodes of different sets or types, object nodes connect to motion nodes, and motion nodes connect to object nodes. Edges are directed to inherently indicate an order or sequence of actions – which is inherently found in cooking recipes or procedures – within the network; hence, it is more accurate to define FOON as a directed acyclic graph because of the direction of edges.

To suitably capture the essence of actions within a FOON, we denote a collection of object nodes and motion nodes describing a single, atomic action within an activity or sequence as a *functional unit*. A functional unit describes the change in the states of objects used in a manipulation action before and after execution; it is important to consider the change in an object's state to identify when an action has been completed [104], which is the primary purpose of structuring actions as functional units in a FOON. Each functional unit contains a single motion node describing the action. Typically, an activity is represented by a series of functional units that are connected by common object nodes. *Input* object nodes describe the required state(s) of objects needed to perform the task, and *output* object nodes describe the outcome of performing the action on those input object nodes. Some actions do not necessarily cause a change in all input objects' states, and so there may be instances where there are fewer output object nodes than inputs. When considering a sequence of actions in an activity, functional units will be connected to one another via overlapping input and output object nodes. A series of functional units connected in this manner, describing an entire activity, is referred to as a *subgraph*; a collection of these subgraphs form what we refer to as a *universal* FOON. Functional units within a FOON are kept unique based on the combination of input/output object nodes and the connected motion node within them.

2.1.1 Network Data Structure

Like typical graphs or networks, a FOON is represented by conventional representations, namely *adjacency matrices* and *adjacency lists*. We use an adjacency matrix to represent a universal FOON for performing network analysis, where each node is represented by a row and its relation to other nodes is given by the columns of the matrix. When performing network analysis, we

present FOON as a one-mode projection; this will be explained further in Chapter 3. An edge from a node N_i to N_j is denoted by a value of 1, preserving directionality of edges; if two nodes are not connected, then an index has a value of 0. Accompanying the adjacency matrix is a *node list*, which keeps track of all object and motion nodes found in the graph. We typically use this structure within the programmed functions of the FOON API, where each node contains a reference to its neighbouring nodes. This provides directional information to other nodes that it is connected to. To create the adjacency matrix programmatically, we use the adjacency list structure to map each node to its row/column representation.

2.2 Accessing FOON

For interested researchers who would like to use FOON in their work, for tasks such as video understanding or task planning, we have made every individual subgraph and a combined universal FOON of 100 videos available for download through our website, which can be accessed at [105]. We also have this universal FOON accessible as a Neo4J graph database, which can be accessed through our web server; users will have to contact our lab members for log-in credentials. Following a study done by a former undergraduate [106], Sanjeeth Bhat, he determined that the graph-based database structure was more appropriate for storing FOON. Additionally, we have provided an API that operates on FOON, which is written in both Python and Java and uploaded to BitBucket for convenience [107]. This API allows users to perform many operations on FOON subgraphs, including but not limited to: reading subgraphs in the form of text files, generating reports on the different node types (object or motion) found within a FOON, performing network centrality analysis on FOON, performing task tree retrieval on FOON, and merging subgraphs together. In Appendix A, I present an overview of how the FOON API can be used for tasks such as graph analysis and task tree retrieval. On our website [105], we have two simpler tools available for use: one for performing task tree retrieval, and the other for visualizing FOON subgraphs.

2.3 Creating a Universal FOON

We will now discuss how FOON graphs are constructed via annotation and then merged together to form a single network – a universal FOON. The consolidation of many subgraphs into a universal FOON is done via a union operation.

2.3.1 Annotating FOON Graphs

Ideally, a subgraph is annotated automatically through the joint task of object and motion recognition on a video demonstration of a single activity. However, up until this point, our group has instead manually annotated several videos from different sources such as YouTube, Activity-Net [108], and EPIC-KITCHENS [109]. Originally in [2], we have collected 60 YouTube videos for annotation; in [103], we increased the number of videos to 65. Presently, we have a total of 100 video demonstrations annotated for FOON, with an additional 18 videos from Activity-Net, 7 videos from EPIC-KITCHENS, and 10 additional videos from YouTube. These numbers gradually increase as more subgraphs are continuously being generated and added, especially from the aforementioned data sets. In Chapter 3, we provide more details about the network created using these videos and give statistics on the size and composition of *FOON-100*. A low resolution compressed visualization of the generated universal FOON is shown as Figure 2.2.

Since subgraphs are created separately by different volunteers, they are often prone to inconsistencies in labels used within functional units. Therefore, a parsing script was developed to preprocess all labelled subgraph files to keep all labelling consistent. This script has three main functions: 1) to create a main index with a list of all the objects, states and motions, 2) to update the input file by relabelling the nodes so they are consistent throughout, and 3) to create a records file that records all changes in any modified files. To keep track of all data elements, the records file contains the object name, its old identifier, its new identifier, the object's initial state, the object's final state, the file name, and the motion names for each functional unit. The parser can also find possible duplicates in objects or motions through the use of the WordNet lexical database [86] by comparing the stem word with the current object index. The index files and the parsing script are provided for researchers in [107] in case new subgraphs are required.

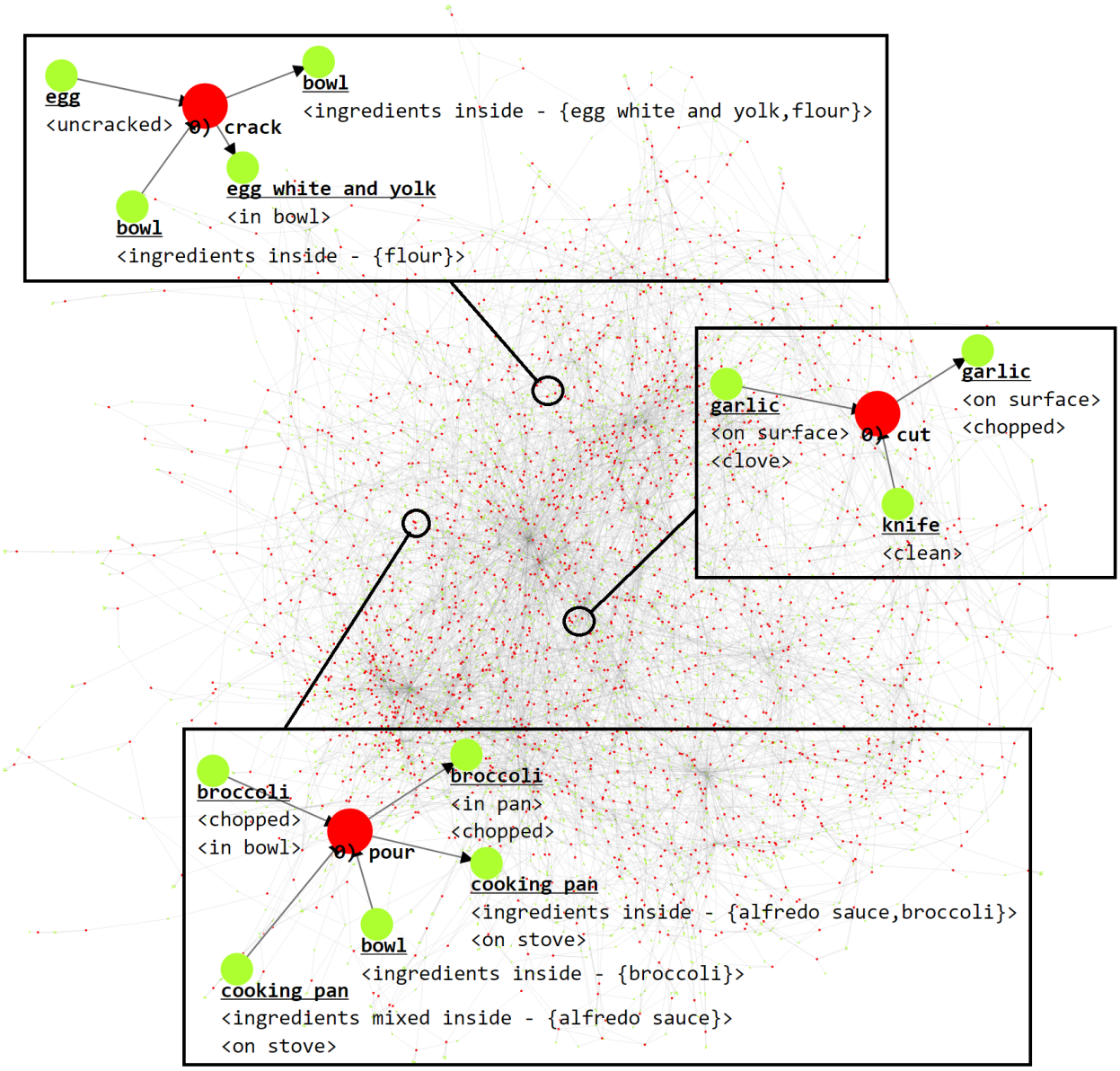


Figure 2.2. Illustration of a universal FOON combining knowledge from 100 instructional videos. Three examples of functional units are shown, each describing an atomic manipulation.

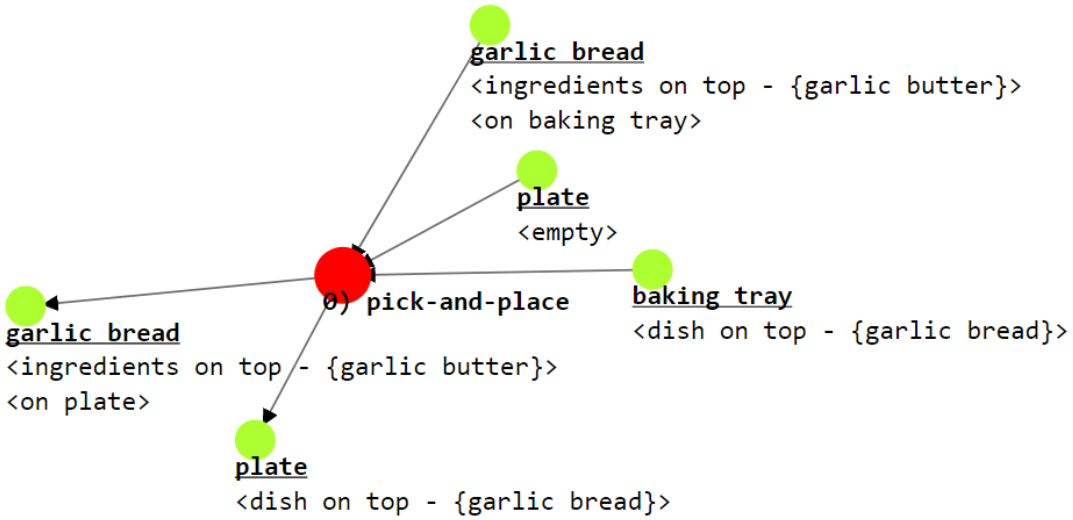
The rules for annotating these graphs are as follows:

- We begin by watching the video. While watching, we internally segment the video and note the actions or steps that are happening in the demonstrated activity.
- While watching the video, we note the objects that are being used in the observed actions. In particular, we note the *states* in which the objects are in, and we also indicate whether those corresponding states are observed before or after the action occurs.
- For each motion observed in the activity, we note the *time-stamp* describing where in the video the action occurs. We record this by noting the start and end time for the action.
- Finally, we also identify which objects are either *actively* or *passively* manipulated in the action. Active manipulation (indicated with a label "1") means that the object or tool is primarily manipulated and used within the action, and it acts upon other objects that are considered to be passive (indicated with a label "0").

In Figure 2.3, we give an example of how a functional unit is written textually.

2.3.2 Merging Subgraphs

A FOON can be used by robots as a knowledge base that is referenced when solving manipulation problems in the household. Such a source should contain a wide array of information from several manipulation demonstrations to achieve specific goals or outcomes. Hence, a universal FOON is ideally constructed from merging many FOON subgraphs. The merging procedure draws new connections (i.e. edges) between objects and new motion nodes as functional units. After the nodes are made consistent within all the subgraphs, we run the union operation to merge all subgraphs into a universal FOON graph – one at a time. Pseudocode describing the merging procedure is presented as Algorithm 1. An example of the merging procedure is provided as Figures 2.4 and 2.5. In detail, the universal FOON, which is denoted as G_{FOON}, is first initialized as an empty list. Then, for each subgraph that we are merging with G_{FOON}, we iterate through each functional unit FU contained within it. For each functional unit we are trying to add to G_{FOON}



(a) Diagram of functional unit

```
O13  baking tray  0
S49  dish on top        {garlic bread}
O229 garlic bread 0
S146 ingredients on top {garlic butter}
S170 on baking tray
O405 plate        0
S54  empty
M38  pick-and-place Assumed Assumed
O405 plate        0
S49  dish on top         {garlic bread}
O229 garlic bread 0
S146 ingredients on top  {garlic butter}
S190 on plate
//
```

(b) Text equivalent of functional unit

Figure 2.3. An illustration of a functional unit (Figure 2.3a) and its textual equivalent as a file (Figure 2.3b). In Figure 2.3b, the numbers refer to integer identifiers given to each object, motion, and state label. We provide index files for each type along with our FOON data set.

Algorithm 1: Merging functional unit FU_{new} with G_{FOON}

Let FU_{new} be functional unit to add
{*Determine whether functional unit already exists in universal FOON:*}
found = False

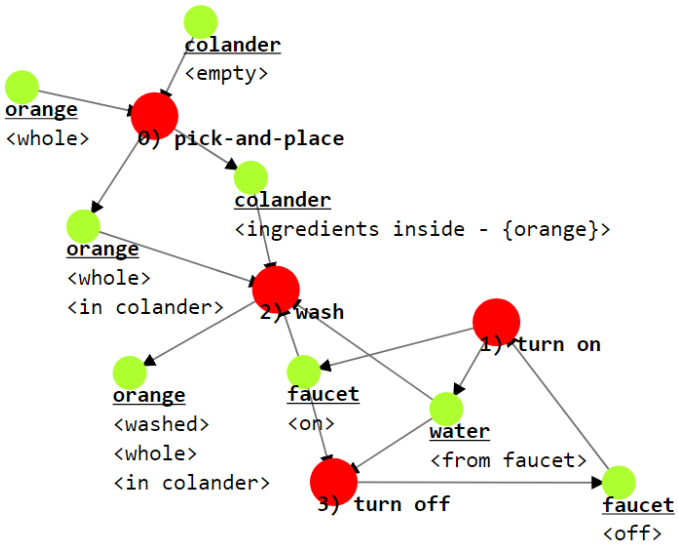
for all functional unit FU_i in G_{FOON} **do**
  if $FU_i == FU_{new}$ **then**
    found = True
  end if
end for

{*If we found the functional unit, we do not add it:*}
if found == $False$ **then**
  Add FU_{new} to G_{FOON}
  {*We must also add the nodes from the functional unit to universal list of nodes:*}
  Add input nodes N_{Input} to node list N_{FOON}
  Add output nodes N_{Output} to node list N_{FOON}
  Add motion node N_{Motion} to node list N_{FOON}
end if

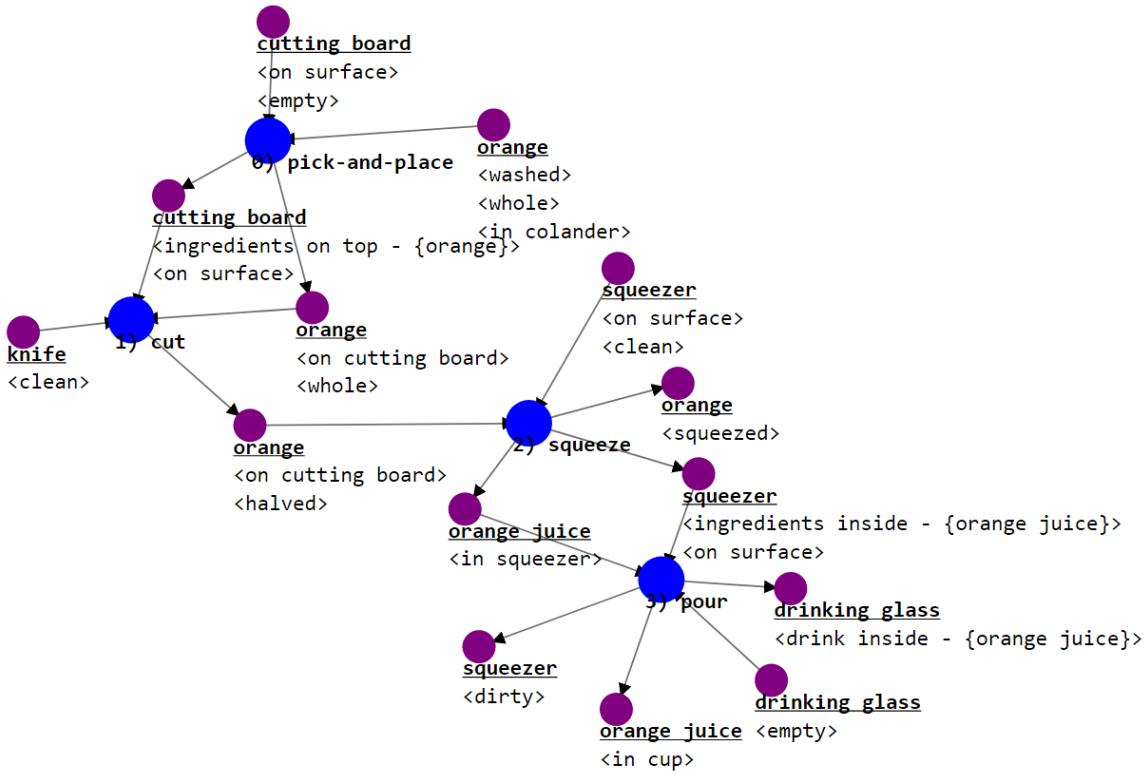
(denoted as FU_{new}), we determine whether a copy of that functional unit is already present in FOON. Duplicates among functional units are indicated by overlap, which is suggested by all of the following conditions: 1) both functional units have the same number of input/output objects, 2) in both units, every input/output object has a matching node of the same object-state combination, and 3) both functional units have the same motion node type. If the unit FU_{new} is deemed to be unique, then it is added to G_{FOON}. Objects in the newly added unit are also added if they did not exist in the universal FOON's node list; however, if they exist, a reference is made to those existing nodes and then the edges are connected to a new motion node. The time complexity for this union operation would simply reduce to the total number of functional units being added to FOON, i.e. $O(|FU|)$.

2.4 Abstraction of FOON using Levels of Hierarchy

Following our previous work in [2], we developed a new way of presenting knowledge in our graph as different levels of abstraction. We can condense the information presented as a universal FOON in an abstracted way through the use of *hierarchies*. Here, abstraction means that we want



(a) Subgraph 1: Washing oranges



(b) Subgraph 2: Cutting and squeezing oranges

Figure 2.4. Illustration of two subgraphs which will be merged into a single FOON (best viewed in colour). Using subgraphs as Figures 2.4a and 2.4b, we create a single, merged procedure of making orange juice from scratch, shown as Figure 2.5.

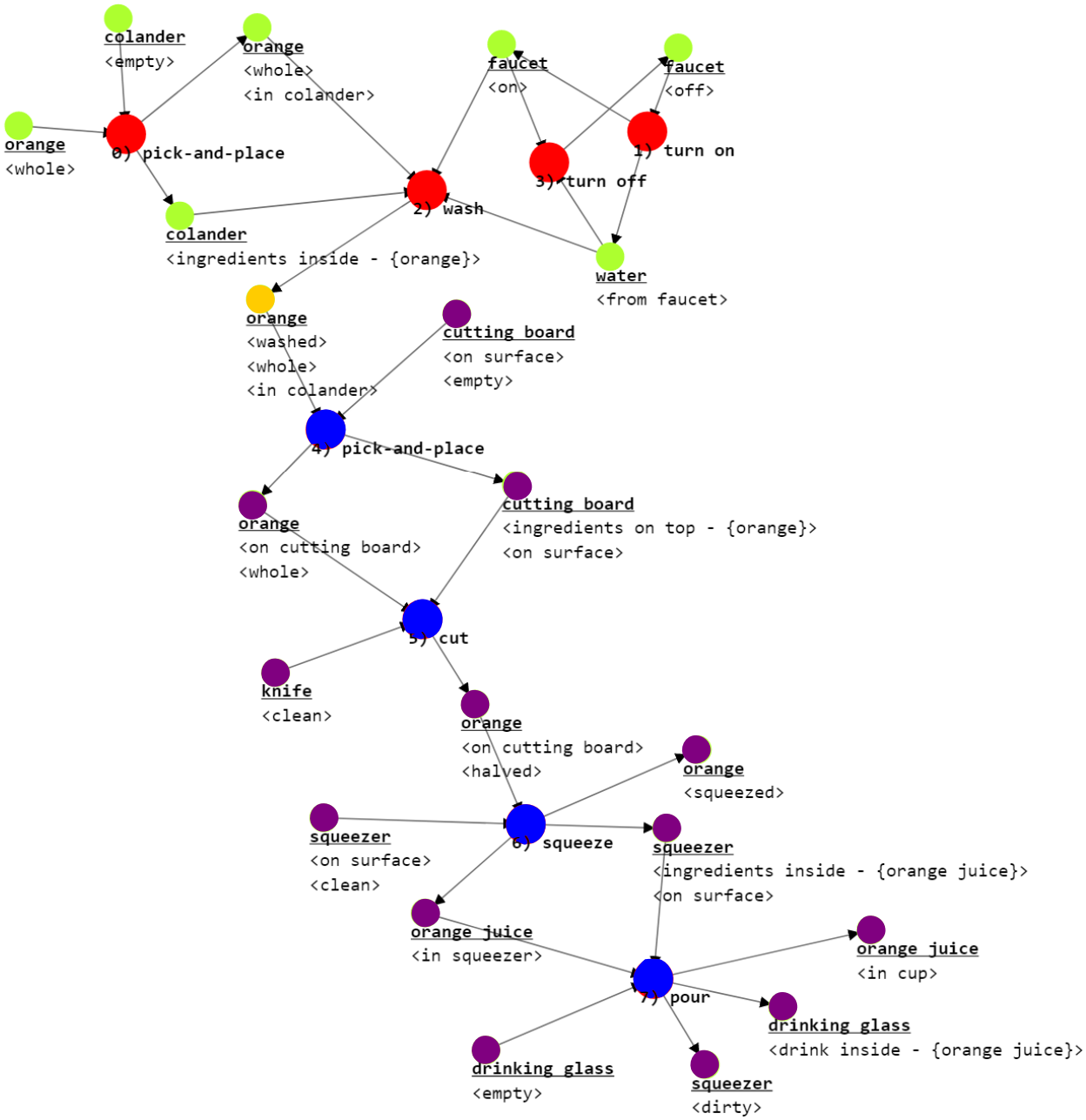


Figure 2.5. Illustration of merged FOON created from combining Figures 2.4a and 2.4b (best viewed in colour). Nodes retain their original colours except for a single node *orange*, which is aptly coloured in orange.

to consider objects in a FOON with as few details as possible; more specifically, there may be times when we wish not to consider or cannot work an object's state or contained ingredients. Hierarchies are useful when we do not require as much detail for performing manipulation tasks; for instance, an object recognition system may not be built to detect certain objects in a variety of states, and so a robot can refer to a version of FOON which does not take states into account. The lower the hierarchy level, the less information is given to object nodes in functional units and the fewer functional units in a FOON (due to there being fewer instances of duplicate units). We can show object nodes at the following three levels of abstraction:

- Level 1 – A FOON at the *purely object* level:
  - Objects are described without any states or contained ingredients. For example, a *mango* object in the *peeled* state and another instance in the *chopped* state are represented by a single *mango* object node in a level 1 FOON.
- Level 2 – A FOON at the *object-state* level:
  - Objects are only differentiated based on their states. Following the example above, we would have two separate object nodes to denote the above objects since there are two instances of *mango* objects with different states *chopped* and *peeled*.
  - However, if we have two objects with mixtures like a *bowl* with *eggs and salt* and a *bowl* of *eggs, milk, salt,* and *pepper*, we treat them as one object node in a FOON: a bowl with *ingredients mixed inside*.
- Level 3 – A FOON at the *object-state-content* level:
  - Objects in different states are classified as unique, separate nodes if the ingredients that make up that object-state node differ to other node instances.

2.5 Future Work: Considerations for FOON Structure

FOON captures the essential components for typical manipulations: objects, ingredients or tools and the type of manipulation executed. However, there may be some elements that could make the representation more complete for robotic manipulation and video understanding tasks.

2.5.1 Considering Multiple States of Objects

To align with preliminary work that has been done on state recognition [104], FOON will be modified to include states that have been suggested as ideal state classes for generalization. With respect to adding location information to nodes in FOON, although the FOON API has been revised to account for annotations of objects with multiple states and locations, our latest version of FOON does not contain information on the object's locations as seen in demonstrations. However, FOON could be integrated with a reasoning system that can infer the location of certain objects, and a robot must rely on its vision system to identify what the objects are in the environment. This concept has been explored in other representations such as SOM [41, 42].

2.5.2 Recipe Look-up from FOON

When considering recipes, however, this is not enough for cooking; in recipes, for example, they outline the quantities or portion sizes of ingredients needed for preparing meals. FOON at the moment has no sense of quantity or portion sizes, which is very important in measuring ingredients for cooking. However, since several sequences share common intermediary actions (i.e. among various subgraphs, there may be common functional units), it is not trivial to assign measurements to ingredient object nodes in FOON, as portion sizes are dependent on what is being made. We cannot arbitrarily assign such values to nodes before run-time as well. Thus, each demonstration in FOON could be encoded with typical portion sizes as it pertains to the various possible end goal items, and serving sizes should be used to calculate the required amount of ingredients needed to prepare a meal at run-time. One way this can be done is by maintaining a link between each functional unit to the subgraphs they were found in. Another way is to embed

ratios within all end-nodes for each recipe used in making a universal FOON; in this way, the merging of units will not affect different amounts across several procedures. Overall, ingredients and other objects in general should be constantly monitored, where a system may be able to report on what items are running low for restocking inventory. Furthermore, portion size monitoring can also enforce healthy eating habits for humans, especially for those who need personalized and regulated diets.

2.5.3 Automatic Creation of FOON Graphs

As of now, our group has been able to achieve semi-automatic construction of FOON graphs directly from video demonstrations [102]. However, more work needs to be done on the recognition of states [104], which itself is a very difficult problem and many research in computer vision do not go into fine-grained object detection that would distinguish states from one another. Presently, we are exploring how we can use natural language processing (NLP) to construct FOON subgraphs from recipes given as text. Existing datasets such as Recipe1M+ [110] are more easily available for researchers to obtain recipes and images that have been parsed and encapsulated into a single source. However, the use of NLP to process textual recipes further highlights other issues such as inferring labels and states that are not explicitly stated in recipe instructions (e.g. the use of utensils or tools, which are rarely provided in the list of items needed in text recipes). Nevertheless, such approaches combined with our preliminary methods of annotating graphs can diminish the time needed for manual annotation and thus should be welcomed and further refined for use on a variety of recipe types.

Chapter 3: Network Analysis of FOON

FOON, just like many large networks across several domains such as social engineering and biology, captures very useful details in its structure. In particular, we are interested in the relationship between objects and the motions observed throughout the network. We primarily focus on determining the most *central* (or important) nodes in our network. The importance of a node innately lies in the frequency of this node's interaction with many other nodes. This measure of importance in network and graph theory is referred to as *centrality*, which is a measured value of importance that is then individually assigned to each node. There are many ways of computing the centrality, and the measures we have applied to FOON were *degree* centrality, *eigenvector* centrality and *Katz* centrality [111]. We use a one-mode projected network specifically for centrality analysis on objects to uncover the relationship between tools and ingredients used in a FOON and in the cooking domain. Using the motion nodes assigned to each functional unit, we can also measure the frequency at which each manipulation motion appears in a universal FOON. We can apply the information obtained to our specific application, where we can determine a set of objects that are frequently used together by the robot from object centrality, and which manipulation skills are the most important for the robot to learn well from motion frequency.

3.1 FOON Statistics

Before our analytical discussion on FOON, we will talk about the size of our universal FOON. We consider three (3) major iterations of FOON throughout the lifetime of our research project, which was introduced in [2], [103] and [112] (tentative) respectively. In [2], our network comprised of 60 videos taken from YouTube. In [103], our network expanded to 65 videos, where the

This chapter was partially published in [2]. Permission is included in Appendix B.

Table 3.1. Statistics of current universal FOON, viz. *FOON-100*.

Hierarchy Level	*# of Object Nodes*	*# of Motion Nodes*	*Total Nodes*
Level 1	420	1594	**2014**
Level 2	1958	1803	**2761**
Level 3	3407	1921	**5328**

additional 5 videos also came from YouTube. FOON has since grown to 100 videos: along with the 65 videos, we have also added 10 new YouTube videos, 18 videos from Activity-Net [108], and 7 from EPIC-KITCHENS [109]. The reason we picked only a small number of EPIC-KITCHENS videos is because the recorded egocentric videos tended to be very lengthy demonstrations, so an entire sequence of preparing a meal can be broken into several videos. Within this work, we will refer to the latest iteration of FOON as *FOON-100*, while the prior iterations will be referred to as *FOON-60* and *FOON-65* respectively. Due to the addition of 35 videos, along with the recent iteration of FOON, the number of nodes in our universal FOON have almost doubled in size when compared to *FOON-65*. Among all of the iterations of FOON, *FOON-100* is the sole version that contains objects with multiple states. We present the totals (as well as the breakdown of object and motion nodes) in Table 3.1 for each level of FOON. A visualization of *FOON-100* can be found at our project's website [105].

3.2 Object Centrality

To analyze objects within a universal FOON, we first transform it into a *one-mode projected network*. This method is used to present a bipartite network into a representation that only contains a single type of node. A good reference for network or graph theory to learn more about this and other concepts is written by Mark Newman [111]. As a result, the one-mode projection will remove all intermediary motion nodes and thus produce a graph where object nodes are directly connected to one another; in this way, we can investigate object-object relationships, as objects will be connected if they are related based on association. Objects are connected to each other based on the direction of edges in all functional units in FOON; we illustrate this concept in Figure 3.1. With this network, we can then perform centrality analysis, where *centrality* is a concept referring to the

importance of a node in a graph. Centrality is reflected by computed values that are assigned to each node; object centrality values are not necessary to be integer values, as the computations can involve more than simply counting the node's degree.

3.2.1 Measuring Centrality

Calculating centrality of nodes in FOON allows us to identify those which are very important in cooking activities. The naive metric of centrality which we may easily adopt is *degree centrality*, where we consider the in- or out-degree of nodes to determine importance; for identifying important objects, we care more about what objects are very important for cooking as input nodes, therefore we prioritize out-degree for all object nodes. However, we should not only be concerned with the degree of each node, but we also ought to consider the influence of all nodes connected to every other node. To capture this, we can use a better-suited centrality metric known as *Katz centrality*, which relies on computing the principal eigenvalue of a graph's adjacency matrix. This metric computes a centrality value for each node which is not necessary to be an integer number like degree centrality. Unlike its original counterpart, the *eigenvector centrality* metric, this method is suited for directed acylic graphs since there would not be a zero centrality problem. Katz centrality is defined as follows: let x_i be the centrality value assigned to a given node i and A_{ij} the adjacency matrix entry for node i and j. A parameter α is added to govern the balance between the eigenvector term and the constant term; this value is typically selected to be no greater than $1/\kappa_1$, where κ_1 being the largest eigenvalue of the entire adjacency matrix A. β is an extra value which we add to all nodes such that they will not be given zero centrality; we assume that β is equal to 1, which gives every node, by default, a minimum centrality value of 1.

The Katz centrality value for node i is defined as:

$$x_i = \alpha \sum_k A_{ik} x_j + \beta \tag{3.1}$$

This equation can be rewritten to acquire a vector of centrality values. Let $\mathbf{A}$ be equal to the adjacency matrix of the entire graph, and $\mathbf{I}$ be an identity matrix and $\mathbf{1}$ a vector with all ones. The

following equation is a revised version of calculating the Katz centrality of all nodes (represented as a vector of all centralities $\mathbf{x}$) where :

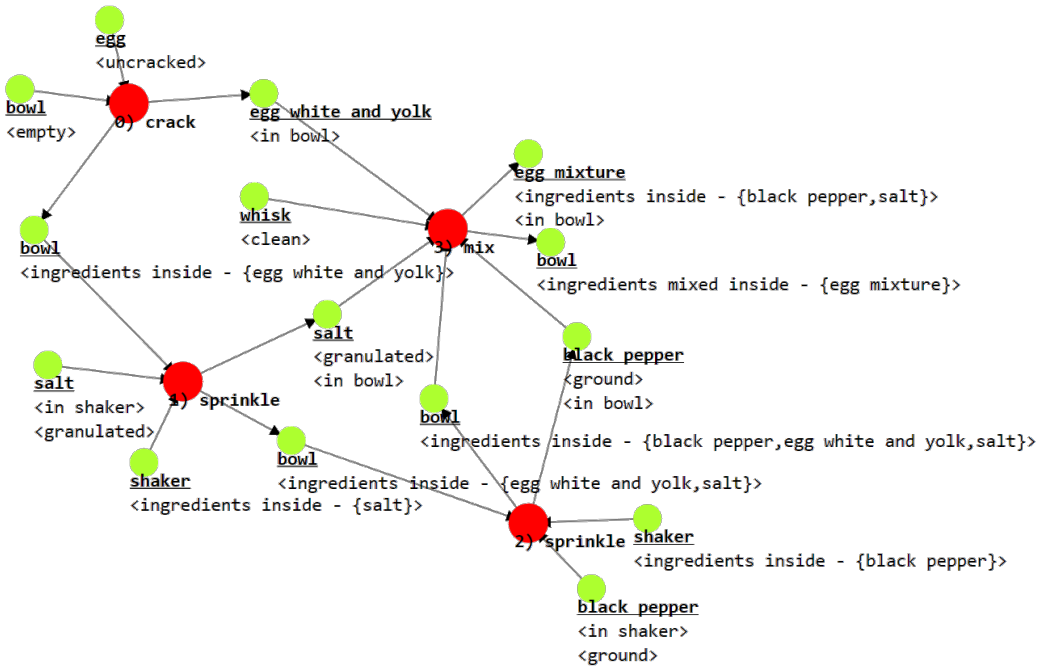
$$\mathbf{x} = \beta(\mathbf{I} - \alpha\mathbf{A})^{-1}.\mathbf{1} \tag{3.2}$$

$$= (\mathbf{I} - \alpha\mathbf{A})^{-1}.\mathbf{1} \tag{3.3}$$

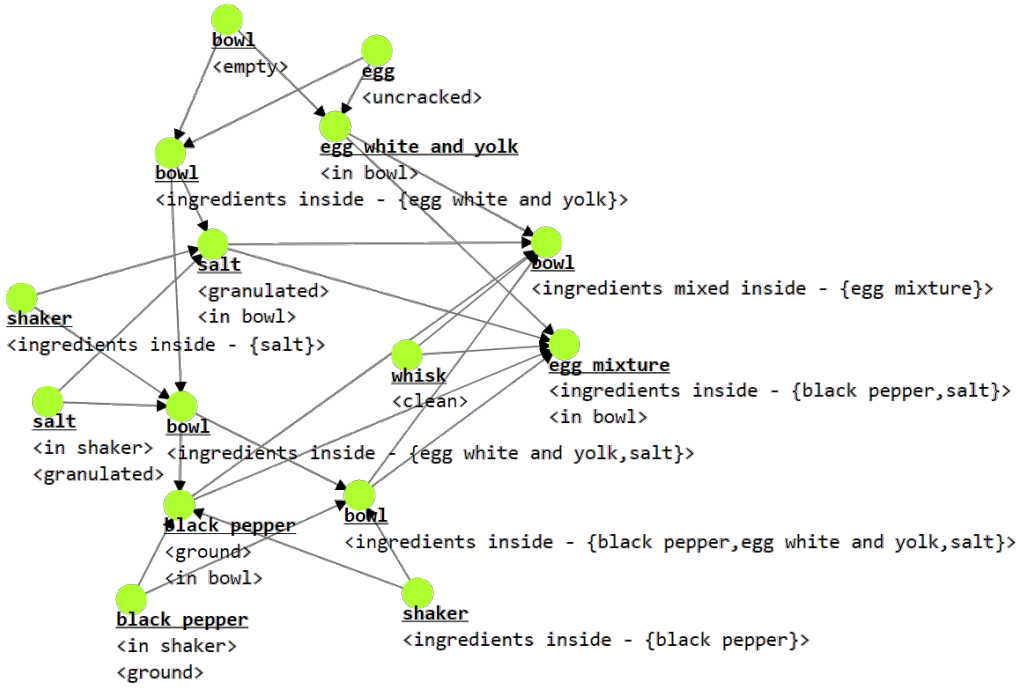
To derive the Katz centrality values for a one-mode projection of FOON, it is important to use the tranpose of its adjacency matrix A, where A_{ij} is equal to 1 if there exists an edge from node i to j and 0 if there is none, rather than using the conventional adjacency matrix form, such that the centrality values give higher values to input nodes.

3.2.2 Centrality Results

The results of object centrality match one's expectations of important objects in the cooking domain. We illustrate the top 20 most central nodes in *FOON-100* as Figures 3.2 and 3.3, showing results for both level 1 and 3 respectively. In these graphs, container objects are highlighted in dark blue, utensils or tools in red, ingredients in gold, and appliances in green. In level 1, several objects overlapped among the top 20 nodes, with a majority of them being container objects. The most important node in level 1 is the bowl object node (with an out-degree of 241), which is followed by the spoon and pan object nodes. In level 3, the majority of objects among the top 20 nodes were utensils with respect to degree centrality but containers with respect to Katz centrality. The most important node in level 3 is a *clean* spoon, (which has a out-degree of 308), which is followed by an *empty* bowl and a *clean* knife. This matches the reality of cooking, as bowls, knives and spoons are very frequently used; spoons and knifes in particular are central utensils that are needed for important tasks of pouring and cutting/chopping. Surprisingly, in the top 20 Katz centrality values for level 3, we see spoon in the state *dirty*; this object was given a relatively high centrality value due to the instance of the action of washing it to obtain a *clean* spoon. Since it will be connected to a *clean* spoon in the one-mode projection, its centrality will also be high as a result of having some relation to the most important node.



(a) Bipartite version of subgraph (original)



(b) One-mode projection of subgraph

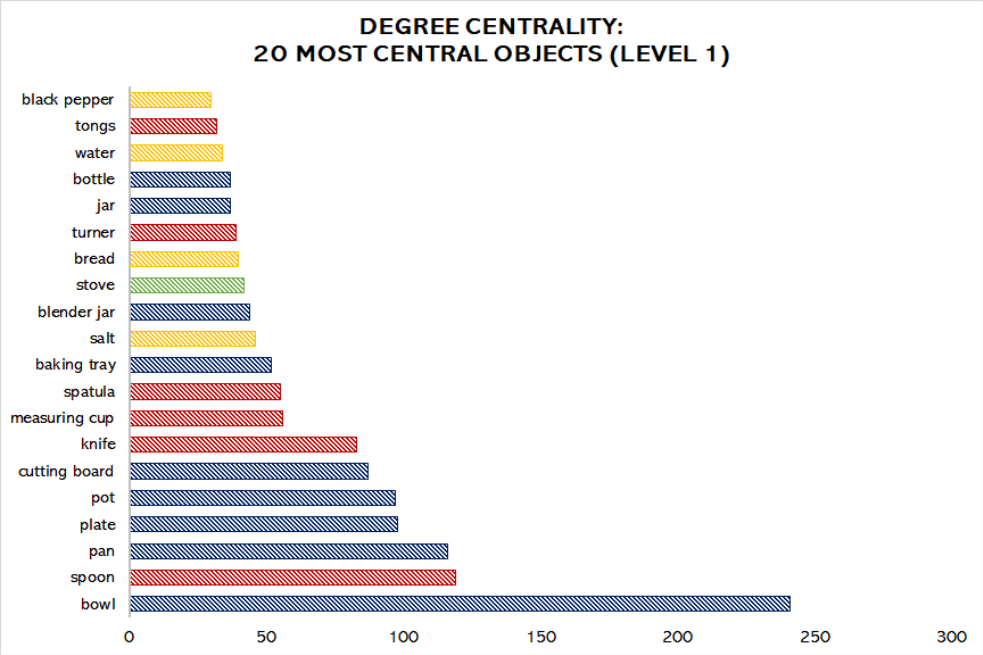
Figure 3.1. Illustration showing how one-mode projection works on an example subgraph (before as Figure 3.1a and after as Figure 3.1b). A one-mode projection of FOON is used for measuring centrality. Input object nodes from each functional unit will directly connect to output object nodes in the same unit.

Centrality values can be used to determine the objects that require the most attention in skill mastery in manipulating them. They also inform which objects are in high demand in recipes across the entire network; as scientists who work among a robot's environment, it is important for us to know what objects are important so that we ensure that these objects are especially made available to the robot and perhaps adapted to the robot's grippers.

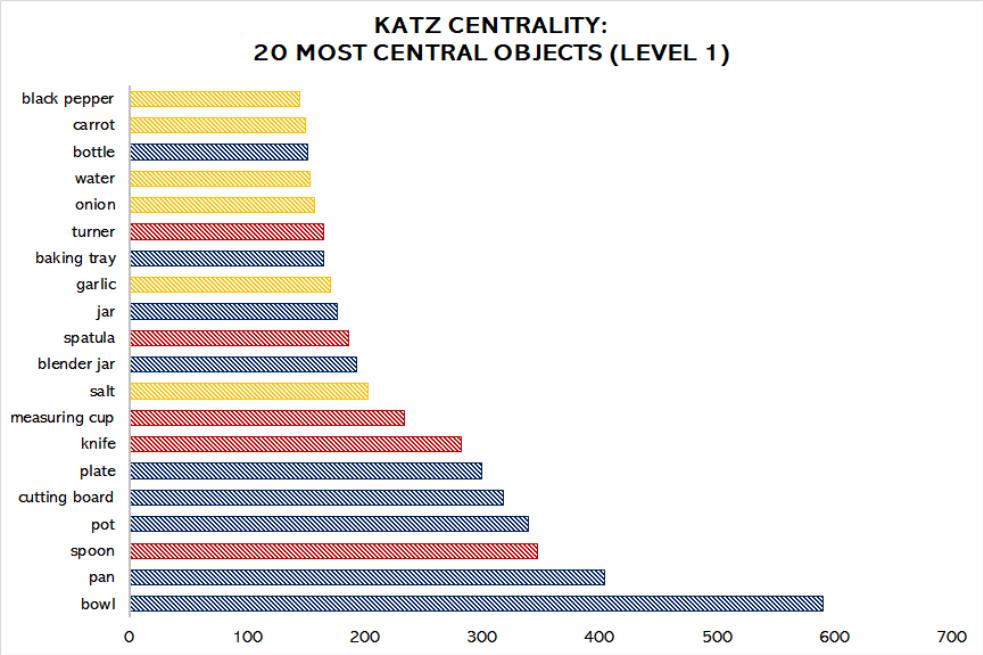
3.3 Motion Frequency

We also consider the frequency at which motions appear as functional units in our network, which we can use for determining the most likely action to occur at a given time and with a given object. We do this by counting the number of instances of each motion belonging to a functional unit that were found in the network. In Figure 3.4, we present the top 20 most frequent motions that are found in *FOON-100*; these motion nodes comprise of 85% of all motion nodes in the universal FOON. The most frequent motion observed (out of 73 possible motion types) is *pouring*. This is a reasonable finding since cooking primarily involves the transfer of ingredients from one container or receptacle to the next until we have our finished product or meal. Second to pouring is *picking-and-placing*, which is merely the act of moving an object from one place to another. This motion is expected to be in the majority; pouring can be seen as a variant of picking-and-placing. Using these motion statistics, we can determine a roadmap for learning manipulation motions based on their prevalence (and thus importance) in FOON. In other words, we should ensure that a robot can masterfully execute the most important motions.

The motion frequency values can also be used in some way to create a probabilistic graphical model from FOON. With these probabilities, we hope to improve our structure to behave more like a typical probabilistic graphical model within the next phases of our project. The frequencies can be used for compressing FOON even further by possibly removing the need for duplicate motion nodes. When paired with the objects, our system would be able to determine the next likely outcome for each object and thus making robot manipulations easier to perform.

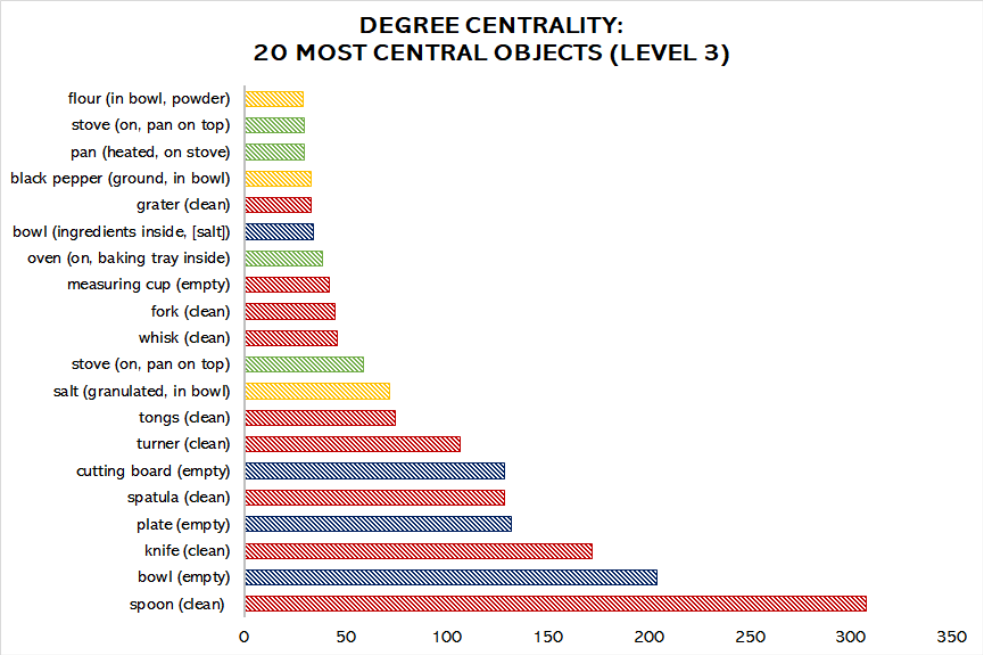


(a) Degree centrality

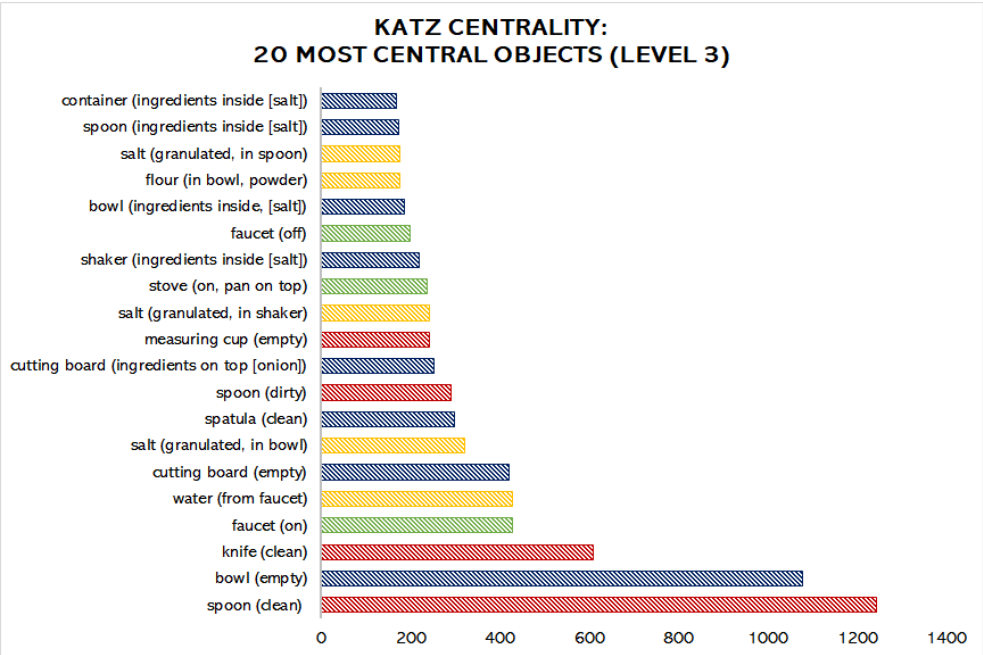


(b) Katz centrality

Figure 3.2. Illustration of both degree centrality (Figure 3.2a) and Katz centrality (Figure 3.2b) for a level 1 one-mode projection of FOON. The colours of each label in these graphs reflect whether it is a container (dark blue), utensil or tool (red), ingredient (gold) or appliance (green).



(a) Degree centrality



(b) Katz centrality

Figure 3.3. Illustration of both degree centrality (Figure 3.3a) and Katz centrality (Figure 3.3b) for a level 3 one-mode projection of FOON. The colours of each label in these graphs reflect whether it is a container (dark blue), utensil or tool (red), ingredient (gold) or appliance (green).

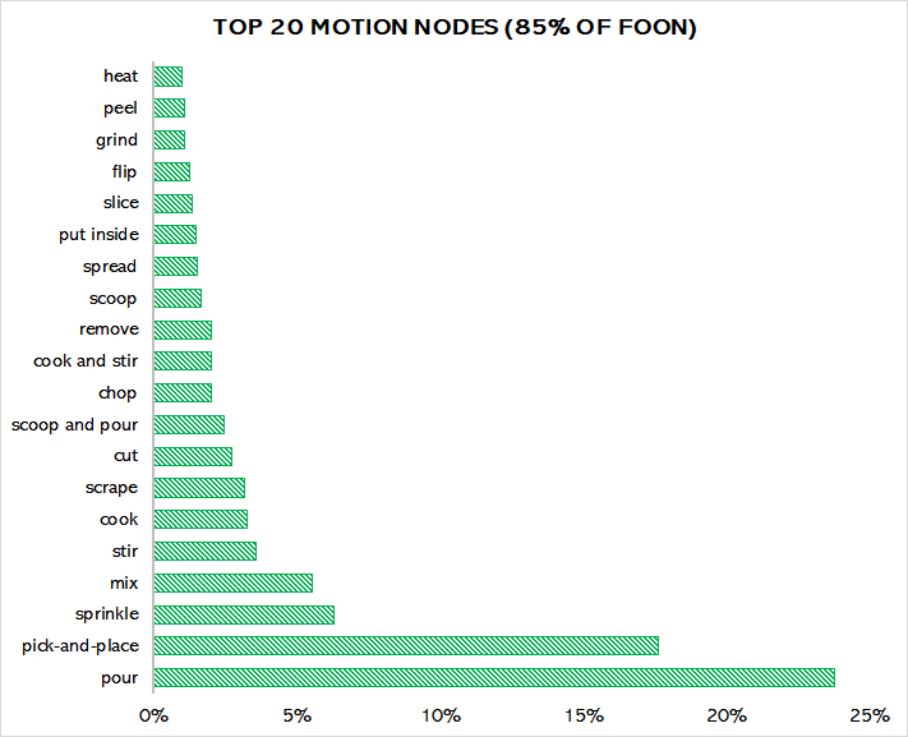


Figure 3.4. Twenty (20) most frequent motions (85% of FOON motion nodes). The top motion labels correspond to the most important actions needed for cooking (aside from regular picking-and-placing), such as pouring, mixing/stirring, and sprinkling.

3.3.1 Motion Aliases

Many of these motion labels have some sort of functional overlap; for instance, scoop and pour can be regarded as a specialized case of picking-and-placing, where scooping is the picking portion and pouring is placing. In addition to pour, sprinkle can also be considered as the act of placing. Other labels are specific variations of existing labels, viz. put inside, remove or take out can be generalized to pick-and-place. In Table 3.2, we list some other examples of motion aliases or "super" classes of motion verbs that may encompass other commonly used motion labels. In Chapter 4, we discuss a proposed method of representing motions that will consolidate such aliases of motions to: 1) compress motion labels (in FOON and in classification algorithms) that have some similarities in functionality, 2) deviates from the sense of human language, thus avoiding ambiguity in natural language, and 3) represent motions in an embedded space, where we can meaningfully define metrics to compute distances (or, in other words, to differentiate) between motions.

Table 3.2. Examples of motion aliases, which were taken from various knowledge sources such as our FOON, MPII Cooking Activities Dataset [113] and EPIC-KITCHENS [109]. With so many verbs that can have multiple meanings, it can be hard to translate between data sets.

Motion Alias	***Equivalent Motion Types***
pick	scoop, uncover (lid), take, take out
place	sprinkle, pour, cover (lid), put, insert, throw (in, away)
pick-and-place	pick-and-place, move, take out, remove, empty,
shake	shake, sprinkle, spice
insert	poke
mix	beat, mix, stir, whisk
cook	cook, bake, heat, blend, fry
cut	chop, cut, dice, slice, cut off ends
insert	insert, pierce, pick-and-place
rotate	twist, turn on (knob), adjust (knob), screw
press	turn on (button), adjust (button)
deform	kneed, squeeze, roll, shape, tuck ends, fold, wrap, take apart, pull apart, rip open

Chapter 4: A Motion Taxonomy for Manipulation Embedding

In learning manipulations from demonstration, the notion of motions is central for task planning and execution for robots; in our FOON representation, motions are pivotal to building functional units for capturing activities. In learning to recognize manipulations in activities of daily living (ADL), it is important to properly define common motions or actions, as representation must be considered for generalization and for a deeper understanding of actions [3]. However, it is very difficult to appropriately define or describe motions – which we understand in human language using words – in a way that is understood by robots. Even with human language, there may be many ways of describing actions, and there are no set conventions or standards for labels to be used for motion recognition and understanding. One major issue in learning motions stems from the need to define a proper representation of these motions, whether it is in activity understanding or motion generation, that appropriately explains the difference between motion types.

In this chapter, we introduce a *motion taxonomy*, which researchers can use as reference to represent motions as a binary string code. The objective of this taxonomy is to derive a descriptive representation of motions from the point of view of the robot by considering the mechanics behind manipulations for measuring distances. These binary strings, which we refer to as *manipulation* or *motion codes*, when used as word embedding can be used in measuring similarity (or dissimilarity) between other motions. With suitable distance metrics, motion classifiers can better discern between motion types or, in the event of uncertainty, suggest similar yet accurate labels for activity understanding. Motion codes can be used with FOON to facilitate the translation of activity descriptions from other data sets and to consolidate variations of motions that have some sort of overlap due to common characteristics.

This chapter was partially published in [114]. Permission is included in Appendix B.

4.1 Motivation for the Motion Taxonomy

Deriving a representation of motions using the motion taxonomy was partially inspired by our own experiences with annotating labels for robot knowledge. We have observed that among several annotators, inconsistency of labelling and defining motions was prevalent. This happens especially with certain motion types that are hard to discern (such as deciding between the labels 'cut', 'slice' or 'chop'), which requires revisiting all labels given to videos to ensure consistency. Furthermore, this is also a problem encountered when using annotated data sets such as the MPII Cooking Activities Dataset [113] or EPIC-KITCHENS [109] since they may have their own labels that may not overlap with each other. In some cases, labels can be very ambiguous and could be better described when adopting data sets for affordance learning For instance, in EPIC-KITCHENS, one verb class provided is 'adjust', which turns out to encompass several actions such as tapping, poking, pressing or rotating depending on types of switches; another example is the 'insert' class, which encompasses actions such as pouring to picking-and-placing. More examples of this is mentioned in the previous chapter, especially in Table 3.2; therefore, it can be a challenge to translate or adopt annotated data from other data sets (in order to save time and effort) to the convention we have adopted in FOON without manual verification.

To potentially resolve these issues, we propose a representation scheme that deviates from natural language since an effective representation is important for robot learning [3]. Binary-encoded strings called *motion codes* will inherently define the motions based on key traits defined in the taxonomy; these codes represent manipulation in a way that robots can "understand" and use to plan and execute. With such strings, we can consolidate aliases or terms for different motions (even in other languages) since they will be represented in a format that describes the motions on a functional level. Ambiguity in human language labels or classes can be better handled if we represent them based on attributes, especially if these can be automatically obtained from demonstration. Ideally, a neural network (or a collection of networks for a group of attributes) can be developed to output codes for different motion types. It is important to note that the proposed motion taxonomy is not claimed to be the ideal way to represent motions; rather, it can be used

to tentatively reduce the amount of features needed to label motions and to compute meaningful distances between motions.

4.1.1 Conventional Representation of Motions

4.1.1.1 One-hot Encoding

Neural networks used for motion recognition typically require motion labels encoded using one-hot vectors as their representation. One-hot vector encoding is a very simplified representation that typically creates vectors of size $1 \times N$, where N is the number of classes. Each vector contains zeroes (0s) except for a row that will contain a value of 1 that maps to a class type; for instance, if we have three labels 'pour', 'sprinkle', and 'cut' given for three motion classes, these may be encoded with vectors $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$ respectively. When training these networks, we typically use the cross entropy loss function, which is defined as:

$$L = -\sum_{k=1}^{N} \mathbf{x}_k \log \hat{\mathbf{x}}_k,$$

where N is the total number of classes (as above), $\mathbf{x}_k$ is the ground-truth distribution, and $\mathbf{x}_k$ is the predicted distribution. Following the example above, during the prediction stage, we can predict the label for a given manipulation sequence with the highest confidence using this equation. Since cross entropy is used to determine how close predicted distributions are to the actual distribution using one-hot vectors, distances between classes would not matter since one-hot vectors are equidistant from one another. Although we can consider this as a distance metric between probabilities, this does not consider class features that can provide a better label for class instances. Following the prior example, we do not get a sense of similarity between motions: pouring and sprinkling can be considered as closer motions than to cutting in terms of manipulation mechanics; such mechanics are represented using our taxonomy.

4.1.1.2 Word Embedding

Word embedding is another technique to derive a vectorized representation of words from natural language. One such technique that learns this in a neural network-like fashion is Word2Vec,

which was originally proposed by Mikolov et al. [101]. With Word2Vec embeddings, cosine distances between vectors suggest relatedness between word labels, where relatedness is determined by context. These models are trained either using *continuous bag-of-words* (CBOW), *n-grams* or *skip-grams* to identify word pairs that are frequently used or seen together; in this sense, word associations are learned based on proximity of words as well as context. However, these vectors do not explicitly describe in what ways words (or in this case, motions) differ, which is one key purpose of motion codes. Furthermore, since Word2Vec derives vectors for singular words, we also can run into issues when defining variations of motions. For example, pushing a solid or rigid object is mechanically different to pushing a soft object since the object we are pushing changes in shape, but we cannot represent these variations with Word2Vec.

4.1.2 Grasp Taxonomies

Taxonomies for robotics have been proposed solely for identifying or describing grasp types. These grasp taxonomies have been extremely inspirational and useful in robotic grasp planning and analysis. A number of works have defined different grasp taxonomies or grasp types [115, 116, 117, 118, 119, 120, 121, 122] from either video demonstrations or grasping data. Those studies have focused on uncovering more than the dichotomy between power and precision grasps (the two main classes of grasps), and they go deep into the way fingers secure objects contained within the hand. Using grasps, we can identify the type of activity happening in a scene, even if the tool is occluded from view, because the type of the grasp can suggest the type of tool being held or manipulated [16]. To some degree, this relates to the theory of affordance [7], where we can infer the functionality of an object based on properties of the object itself. However, there is a lack of a manipulation motion taxonomy that focuses on the mechanics of motions – specifically trajectory and contact in manipulations. Different from the grasp taxonomy that focuses on the finger kinematics, we prioritize contact and motion trajectory. A mechanics-based manipulation motion taxonomy could help roboticists to consolidate motion aliases, words or expressions of the same or similar motions in terms of mechanics and eventually for motion generation, motion

analysis, and recognition in a similar way to how grasp taxonomies have been useful in defining or describing grasps for planning strategies.

4.2 Motion Taxonomy: Version 1

In [114], we proposed the first iteration of the motion taxonomy. To capture the mechanics of the manipulation motion, we looked at the motion from the following main aspects: contact type, engagement type, and trajectory type. We then added two additional aspects that could be useful for planning: contact duration and manual operation (whether unimanual or bimanual) for finer manipulation details. We define a manipulation to be an interaction between active and passive objects, where an *active object* or *manipulator* (which is usually a tool or utensil) acts upon a *passive object* or *manipulatee*. We combine them into a manipulation code to represent a motion. Figure 4.1 illustrates the manipulation taxonomy described in Table 4.3 as five hierarchical trees. Each manipulation motion will be grouped according to the taxonomy trees and assigned a string of binary manipulation code. In the following subsection, we describe each hierarchical tree in detail.

4.2.1 Motion Attributes

4.2.1.1 Contact Type

We mainly distinguish manipulations as *contact* or *non-contact* motions. Contact motions are those in which there is an interaction between objects, tools or utensils in the demonstration, while non-contact are those in which there is little to no contact. *Contact* motions are those manipulations that involve forces being applied on an object (or a set of objects) where the force is exerted by a tool, utensil or another object. We refer to the tool or utensil as the *active* participant in the motion, while objects being acted upon are referred to as *passive* participants. For instance, a hammer exerts force as repeated single, powerful impacts on a nail for the hammering motion, while a softer force can be observed with motions like mixing liquids in a container or brushing a surface with a brush. In some cases, the robot's hand acts as the active tool in manipulations such as picking-and-placing, squeezing or folding. We can also have a *non-contact* motion type, which will

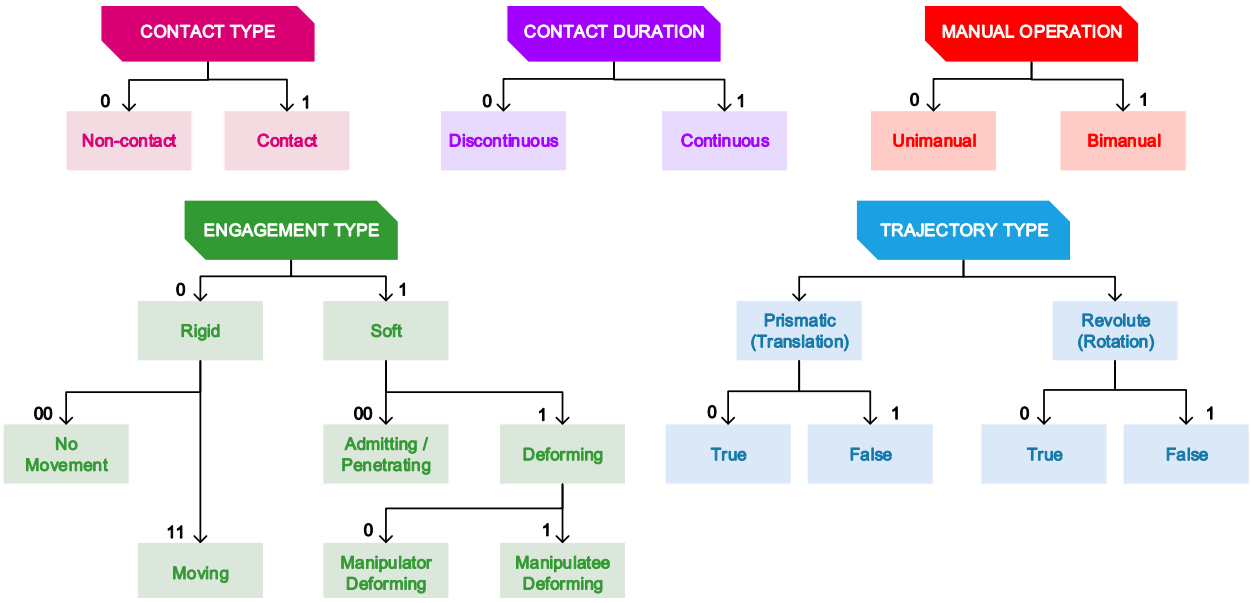


Figure 4.1. An illustration of the first iteration of the motion taxonomy, which was introduced in [114]. This structure has since been revised with additional features, which is shown as Figure 4.2.

involve the manipulation of tools that make little to no contact on participating passive objects. For instance, when we pour a liquid into a bowl from a cup, the cup does not touch the bowl in a typical pouring action. It is important to note that in pouring, we do not consider the hand gripping the object as a tool.

4.2.1.2 *Engagement Type*

A manipulation motion can also be identified by how an active object *engages* with other passive objects. We identify motion engagement types as either being *soft* or *rigid*. *Soft engagement* motions are those where either the active tool or the passive object undergoes a change in its shape from contact with each other. *Rigid* or *neutral engagement* motions have neither the tool nor the objects change in their shape, state or form as a result of direct contact. However, these motions can either cause some sort of *movement* in the manipulation or the object being acted upon does *not move* from the manipulation. For instance, with a spatula, one can pick up items without changing the physical state of the manipulator tool and the manipulated object, but the passive item would be moved from one location to another. Soft engagement contact can be broken down

into three subcategories: 1) *admitting* or *penetrative*, where the tool can penetrate the object without deformation of the tool and the passive object allows the tool to enter it, or 2) *deforming*, where either the active or passive object deforms in some way. The latter can be further broken down into either deforming of the *manipulator*, where the active tool itself changes in its shape or deforms for manipulation upon an object, or deforming of the *manipulatee*, where the passive object changes in its state or shape and the active tool remains rigid and does not deform. As an example of an admitting engagement, when scooping flour from a bowl, the spoon or cup penetrates the ingredients. A manipulator-deforming engagement type can be observed when using a brush, for instance, since the bristles will bend and deform in shape from the default appearance of a brush. As for a manipulatee-deforming engagement such as cutting, the active knife deforms the passive object by changing its shape from its natural state to pieces for the purpose of cooking.

4.2.1.3 Contact Duration

With contact made between the active tool and the passive object, engagement can either be *continuous*, where there is a constant interaction or force in the manipulation over the duration of the action, or *discontinuous*, where there is little to no constant or non-persistent contact between them. Discontinuous motions tend to be those which can be identified by sharp periods of force. For example, in the case of pick-and-place, the only contact between the object and the environment in the pick-and-place process are at the beginning and the end of the process – breaking and establishing contact between the picked object and the support environment. However, since the hand is considered to be the active tool, which continuously grips the object, this action is considered as continuous contact. With an action such as dipping, the object will only make temporary contact with contents usually held within a container.

4.2.1.4 Trajectory Type

As for manipulation motion types, the movement can be *prismatic*, where it undergoes linear translation across a line/plane (e.g. cutting is usually a vertical motion in 1D), or it can be *revolute* or *rotational*, where the object or tool undergoes a change in orientation and it moves about axes

of rotation (e.g. pouring typically involves the rotation of a cup to allow liquid to flow into a receiving container). Manipulation motions are not confined to a single trajectory type since certain manipulations combine rotation and translation; hence, these two subcategories are not mutually exclusive. An example of this type of motion is folding.

4.2.1.5 Manual Operation

Motions can also be described by the number of hands (or end-effectors) regularly used in the action. We can classify them as *unimanual* (involving one hand) or *bimanual* (involving both hands) in terms of manipulation of the active tool or item. Sprinkling salt from a shaker can be considered as a unimanual action since we can hold the shaker and shake it with one hand, while rolling or flattening is usually a bimanual action since a rolling pin requires two hands to operate. This criterion is important for determining which motions we can execute since some robotic systems are not built consistently to human anatomy (i.e. with two arms, two hands, and similar joints).

4.2.2 Manipulation Codes: Version 1

Based on the taxonomy, each motion type can be represented with a *manipulation code* which can be used for representing each motion as detailed in our taxonomy. The binary string is a combination of manipulation attributes in the following order from left to right: contact type, engagement type, trajectory type, contact duration and manual operation. In Table 4.1, we assigned manipulation codes to common cooking motions as seen in both FOON and DIM.

As a result of using the taxonomy, several motions ended up naturally clustered because of common codes. Mixing/stirring is assigned the same code as inserting/piercing since they are both admitting actions, have prismatic trajectories, and they are classified as continuous contact motions. Cutting/slicing/chopping along with motions such as mashing, rolling (unimanual), peeling, shaving, and spreading are clustered together mainly because of their manipulatee-deforming and prismatic properties. This group is separate to the group containing pulling apart and grating because they are typically bimanual actions.

Table 4.1. Manipulation Code (Version 1), which was taken from [114]. Refer to the index in Table 4.3 or Figure 4.1 for the meaning behind binary digits.

Manipulation Code	*Motion Types*
00000100	shake/sprinkle
00001000	rotate, pour
10111000	poke
10111010	pick-and-place, push (rigid)
10111100	flip
11001000	dip
11001010	insert, pierce, mix, stir
11001100	scoop
11101010	brush, wipe, push (deforming)
11110100	tap, crack (egg)
11110111	twist (open/close container)
11111010	cut, slice, chop, mash, roll (unimanual), peel, scrape, shave, spread, squeeze, press, flatten
11111011	roll (bimanual), pull apart, grate
11111110	fold (wrap/unwrap)

4.3 Motion Taxonomy: Version 2

Following our previous iteration, we made several changes to the taxonomy. In our latest version, we consider the following attributes based on contact and trajectory information for the taxonomy: contact interaction type, engagement type, contact duration, trajectory type and motion recurrence. As before, we define a manipulation motion to be any atomic action between *active* and *passive* objects; however, the key distinction of version 2 is that an active object is defined as a hand/gripper, tool or utensil or the *combination* of the hand/gripper and tool that acts upon passive objects. As a result, motion codes have been revised to indicate whether the active object is solely a hand/gripper or if it is a combination of a hand or tool. Additionally, we no longer consider motions to be unimanual or bimanual, since we need not care about the number of hands required for a manipulation if we can program a robot to use more than two hands or grippers. Instead, motion codes can be defined for each end-effector or gripper used in the manipulation that will describe how it operates the tool or how it affects the passive object. When considering contact, we still examine whether objects used in the manipulation make contact with one another and we describe what happens to these objects when this contact is established. These revised

features are shown in an updated hierarchical tree as Figure 4.2, and they are further described in the following subsections.

4.3.1 Describing Contact Type and Features

Motion types can be classified as *contact* or *non-contact* interaction types. Contact motion types are those that require contact between an active object (i.e. the actor's hands or the object that is typically grasped in the actor's hands) and passive object(s) (i.e. the object(s) that is/are manipulated upon when in contact with an active object) in the work space. As opposed to the taxonomy in [114], we may consider the hand or end-effector as a tool. Conversely, non-contact motion types are those where no contact is established between active and passive objects or there is no force exerted upon passive objects. Contact can be observed with vision (for instance, by the objects' borders or bounding boxes overlapping) or using force sensors mounted on objects. An example of a contact motion is mixing, where the active tool makes contact with contents within a passive container. As for a non-contact motion, pouring is a prime example: when pouring from one container to another, the active container held in the hand is not required to make contact with the passive receiving container.

Once an object interacts with others through physical contact, we classify their engagement as either *rigid* or *soft*. Rigid engagement is where an active object's interaction with passive objects does not result in deformation – i.e. their structure is not compromised or changed –, whereas soft engagement is where objects deform as a result of the interaction or the objects allow admittance or are permeable. An example of a soft engagement motion is mixing, as the passive object will allow the active tool to permeate it and alter its structure; on the other hand, an example of a rigid engagement motion is picking-and-placing, as the passive object's structure does not change at all since the active manipulator (i.e. the hand) is simply translating the passive object in the scene.

In addition to the prior attributes, it may be useful to note whether the active tool makes persistent contact with passive objects. If the actor only makes contact for a short duration in the manipulation, we consider that contact to be *discontinuous*; however, if the contact between the active tool and passive object is persistent, we consider that contact to be *continuous*. However,

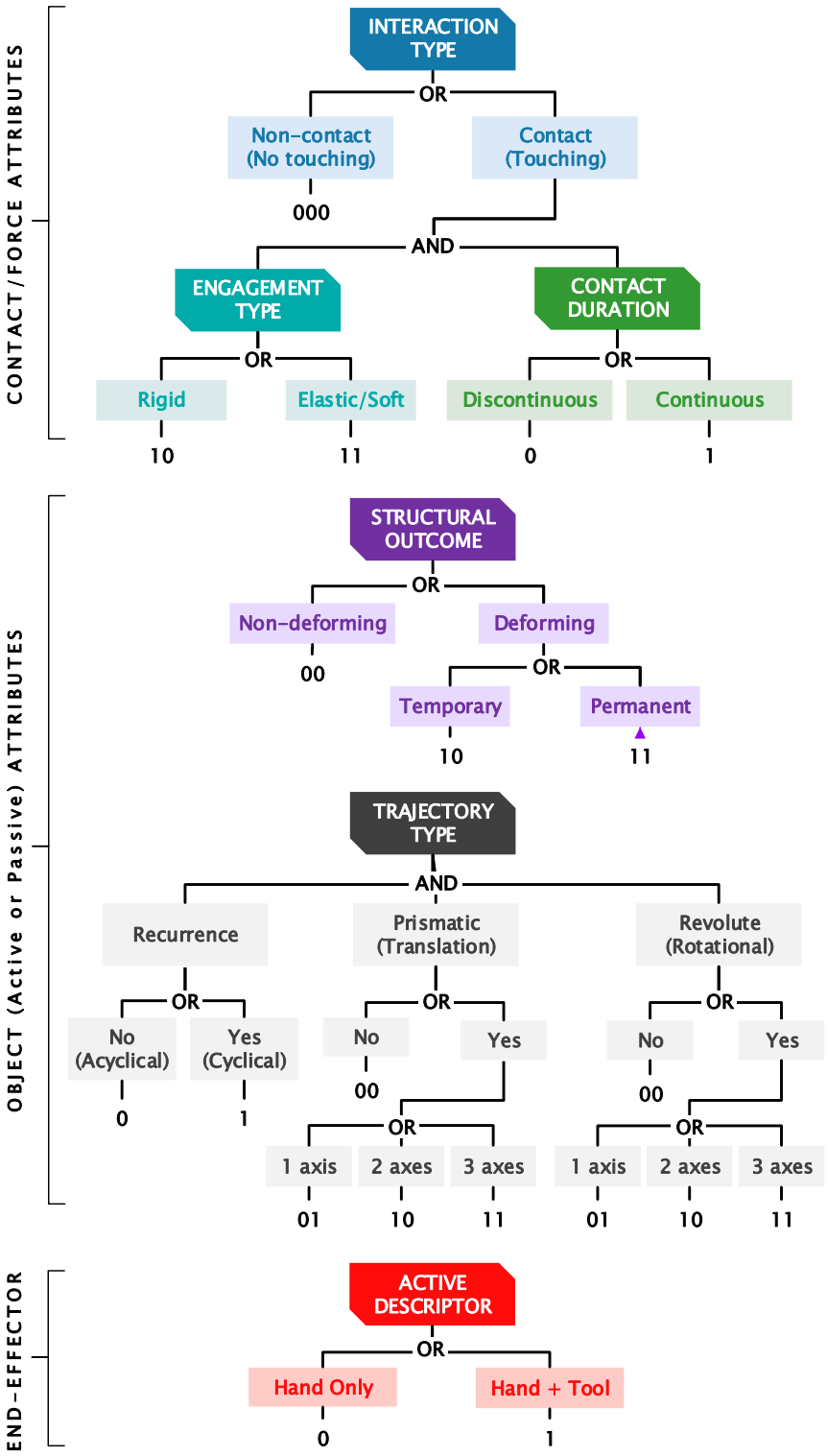


Figure 4.2. An illustration of the motion taxonomy (Version 2). A motion code is formed by appending contact features, the active object's structural bits, the passive object's structural bits, the active trajectory and passive trajectory bits, and active bit descriptor by following the tree.

this perspective changes depending on what is considered to be the active object. If we consider the robot's hand to be the active tool only, then we can assume that once it is grasping a tool for manipulation, there would be continuous contact between the hand and the tool. This is why we consider the active tool to be either the hand (if there is no tool acting upon other objects) or both the hand and tool as a unit (if there are other objects in the manipulation). Contact duration can be determined visually (by timing the overlap of bounding boxes placed over each object participating in the activity, for instance) or physically with sensors or trackers.

4.3.2 Describing Changes in Object Structure

We can also consider the structural integrity of the objects used in order to describe deformation. Active and passive objects can either undergo no deformation (non-deforming) or structural change (deforming). We consider the cutting action as a soft engagement motion, as an active *knife* object will permanently deform the passive object into smaller pieces or units; even in the action of mixing items within a bowl, the contents within the bowl can be regarded as the passive objects being acted upon and deformed. As for a rigid motion, actions such as tapping or poking a solid object show no structural change among objects as a result of contact. In spreading with a knife, neither the knife nor the surface (like bread) incurs a significant change in their shape. Deformation can be further distinguished as temporary or permanent, which is attributed to the material or texture of the objects. For instance, when we squeeze a passive *sponge* object, it returns to its original shape, signifying that this motion temporarily deforms it. However, in the cutting example from before, this state change is permanent. Poking or tapping an object would classify as soft engagement if we were to tap or poke a soft or elastic object, which would typically show a temporary sign of deformation. However, in other cases, it would be regarded as a rigid engagement with no observable structural change.

The combination of these features and the previously discussed contact engagement features were once combined as a single tree in the first version of the taxonomy (refer to Figure 4.1); however, we found that the state of the object is not central to the mechanical properties of either rigid or soft motions since it simply describes what has happened to the objects after the manipulation is

executed. However, with this separation, we are able to describe the structural change for both active and passive objects, as opposed to the prior version, which considers that only one of these entities changes in its state.

4.3.3 Describing Trajectory of Motion

As we have done before, we can describe an object's trajectory as *prismatic* (or translational), *revolute* (or rotational), or *both*. Prismatic motions are manipulations where the object is moved along a certain axis or plane of translation. Prismatic motions can be 1-dimensional (along a single axis), 2-dimensional (confined to a plane) or 3-dimensional (confined to a manifold space); this can be interpreted as having 1 to 3 DOF of translation. Revolute motions, on the other hand, are manipulations where the object is rotated about an axis or plane of rotation; a robot performing such motions would rely on revolute joints to execute manipulations of this nature. Similar to prismatic motions, revolute motions can also range from 1-dimensional to 3-dimensional motion (i.e. from 1 to 3 DOF of rotation); typically, revolute motions are confined to a single axis of rotation in world space. This differs to the trajectory hierarchy found in [114], which solely considers if a motion has prismatic/revolute properties or not. A motion is not limited to one trajectory type, as these properties are not mutually exclusive; therefore, we can say that a motion can be prismatic-only, revolute-only, neither prismatic nor revolute or both prismatic and revolute. From the perspective of the active object, an example of a prismatic-only manipulation is chopping with a knife since the knife's orientation is usually fixed, while an example of a revolute-only motion is fastening a screw into a surface using a screwdriver. However, a motion such as scooping with a spoon will usually require both prismatic and revolute movements to complete the action.

As an addition to the taxonomy, we can also describe a motion's trajectory by its *recurrence*, which describes whether the motion exhibits repetitive behaviour in the tool's movement. A motion can be *acyclical* or *cyclical*, which may be useful depending on the context of motion. This simply answers the question of whether the motion is observed to be repetitive or not, as some actions may require some repetitive motion to finish them and to continue a sequence of execution. For instance, mixing ingredients in a bowl may be repeated until the ingredients have

fully blended together, or in the case of loosening a screw, the screwdriver will be rotated until the screw is completely out of the surface. Learned acyclical motions can be made cyclical simply by repeating them, which is a decision that can be left up to the robot during motion generation if it is not finished with its task or it failed to execute the manipulation successfully.

4.3.4 Translating Motions to Code

We now discuss how motion codes can be assigned to motions using the example of the cutting action. Using the flowchart shown as Figure 4.2, we construct codes in the following manner: first, we ascertain whether the motion is contact or non-contact. In cutting, the active knife object makes contact with the passive object, and so we will follow the contact branch. If the motion was better described as non-contact, then we will start with the string *'000'*. Since there is contact, we then describe the type of engagement between the objects and how long the contact duration is throughout the action. Following our example, the knife cuts through an object and maintains contact with it for the entirety of the manipulation, hence making it a soft engagement (*'11'*) and with continuous contact (*'1'*). After describing contact, we describe the state of the active and passive objects after the manipulation occurs. In our example, the active object does not deform (*'00'*) while the passive object deforms permanently since the knife cuts it into a different state (*'11'*). After describing the structural integrity of the objects, we then describe their trajectories. When cutting an object, the active trajectory is typically a 1D prismatic motion as we swing the knife up and down and without any rotation (*'00100'*), while there is no passive trajectory (*'00000'*), as the passive object is usually immobile. If we are observing repetition in cutting, then we would assign the recurrent bit *'1'* instead of *'0'* in the active trajectory substring. Finally, we indicate whether the active object is solely the hand or hand/tool pair; in our example, we would assign it a bit of *'1'* since we have a hand and knife pairing as an active object. With all of these substrings, we end up with the single motion code *'11100110010000001'*.

We compiled a list of motion labels for several common ADL that can be found across several sources of manipulation data such as EPIC-KITCHENS, MPII Cooking Activities, FOON [103], and Daily Interactive Manipulations (DIM) [1]. In Table 4.2, we show codes assigned to other motion

Table 4.2. Manipulation Codes (Version 2), which is based on the taxonomy illustrated in Figure 4.2. The attributes of each motion correspond to those in source demonstrations. Underlined bits correspond to the active object's features, while overlined bits correspond to the passive object's features. Motion codes are 18 bits long.

Motion Code	*Motion Types*
000000000001000001	pour
000000010100000001	sprinkle
100000000100000000	poke, press (button), tap
101000000000000000	grasp, hold
101000000001000010	open/close (jar), rotate, turn (key, knob), twist
101000000100000001	spread, wipe
101000000100001000	move, push (rigid)
101000000101001010	flip (hand)
101000000101001011	flip (turner, spatula)
101000001000000001	spread, wipe (surface)
101000001000000010	open/close (door)
101000001000010000	move (2D), insert (placing), pick-and-place
101000010001100011	fasten, loosen (screw)
101000010001100010	shake (revolute)
101000010100101000	shake (prismatic)
110001000100000001	dip
110001000101001001	scoop (liquid)
110001100101001001	scoop
110110000100000001	crack (egg)
111000000100000001	insert, pierce
111001000000000000	squeeze (in hand, elastic)
111001000101001010	fold, unwrap, wrap
111001011000000001	beat, mix, stir (liquid)
111001100000000000	squeeze (in hand)
111001100100001000	flatten, press, squeeze, pull apart, peel (hand)
111001100100000001	chop, cut, mash, peel, scrape, shave, slice
111001100100100010	roll
111001101000000001	saw, cut (2D), slice (2D)
111001111000000001	beat, mix, stir
111100000100001001	brush, sweep, spread (brush)
111100001000010001	brush, sweep (surface)
111111000000001001	grate

types from source data sets. Several motions can share the same motion code due to common mechanics, such as cutting and peeling since they are both 1D-prismatic motions that permanently deform the passive objects. We can also account for variations in manipulation; for instance, certain motions like mixing and stirring can either temporarily deform or permanently deform the target passive object, which depends on its state of matter. We can also identify non-recurrent or recurrent variations of motions. It is important to note that motion codes can be assigned to each hand or end-effector used in a manipulation since they are not necessary to perform the same manipulation in the same action. For instance, when chopping items, usually it is necessary to hold the object in place with one hand and then use the knife to chop with the other. Because of this, the structural or state outcome of performing those actions could be extrinsic to the actions; in the aforementioned example, the passive object deforms but it is not directly an outcome of just holding the object. In Table 4.2, we simplify this to the single-handed perspective of performing those actions.

4.4 Evaluation of the Taxonomy

Having understood the taxonomy and identified motion codes for manipulations in ADL, we demonstrate how suitable they are for representing motion labels. In particular, we focus on how motion codes can produce embeddings whose distances are meaningful based on their attributes. In this section, we support our taxonomy by comparing force reading data for different motion types from DIM [1], which provides position/orientation and force sensors for a variety of human activities. DIM is the only data set at the moment that contains contact 6-axis force data of many manipulation motions [123]. The objective here is to match each activity to a motion type and to determine whether the measurements show that certain motion types are alike to other motion types, thus determining whether the clusters from Table 4.1 aligns with real data.

4.4.1 Experiment: Support for Motion Codes from Demonstration Data

Preferably, motion codes are derived directly from demonstration data. Several modalities of data such as trajectory, force, and vision can be used to determine the attributes that best describe

Table 4.3. Mechanical characteristics described by the motion taxonomy. Here, we summarize all of the properties and their definitions as it pertains to manipulations and motion codes.

Attributes	*Description of Motion Attributes*
Interaction Type	• **Non-contact** – there is little to no contact between active and passive objects. • **Contact** – there is contact between the active and passive objects.
Engagement Type	• **Rigid engagement** – active and passive objects in activity do not change in structure from contact. • **Soft engagement** – the manipulation causes change in structure of either tool (active) or objects (passive).
Contact Duration	• **Discontinuous** – if contact is observed, contact between active and passive objects is temporary. • **Continuous** – contact persists between active and passive objects
Structural Integrity (State Change)	• **Non-deforming** – no observable change in structure or shape of objects. • **Deforming** – there is observable change in structure or shape, further classified as: → **Temporary** – deformation is temporary (due to elastic material) → **Permanent** – deformation is irreversible
Trajectory Type	• **Recurrence** – the motion may be continuous or discontinuous → **Acyclical** – no repetition in trajectory of active tool or hand → **Cyclical** – there is repetition in trajectory of active tool or hand • **Prismatic** – movement about a line, plane or surface. → **1 DOF** – the tool's movement is restricted to a line or single axis → **2 DOF** – the tool's movement is restricted to plane or two axes → **3 DOF** – the tool's movement is restricted to surface or three axes • **Revolute** – movement about axes of rotation (change in its orientation) → **1 DOF** – the object is rotated about a single axis → **2 DOF** – the object's movement is rotated about two axes or plane → **3 DOF** – the object is freely rotated about all axes
Active Object Descriptor	• **Hand Only** – active object is hand or gripper • **Hand and Tool** – active object is combination of hand/gripper and tool/utensil/etc.

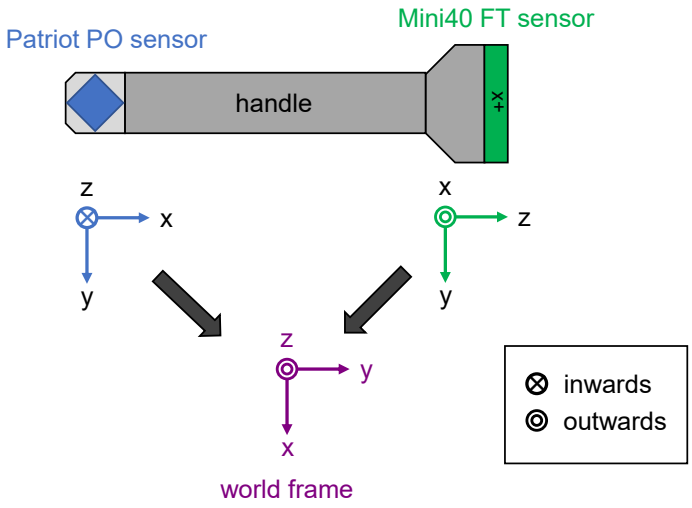
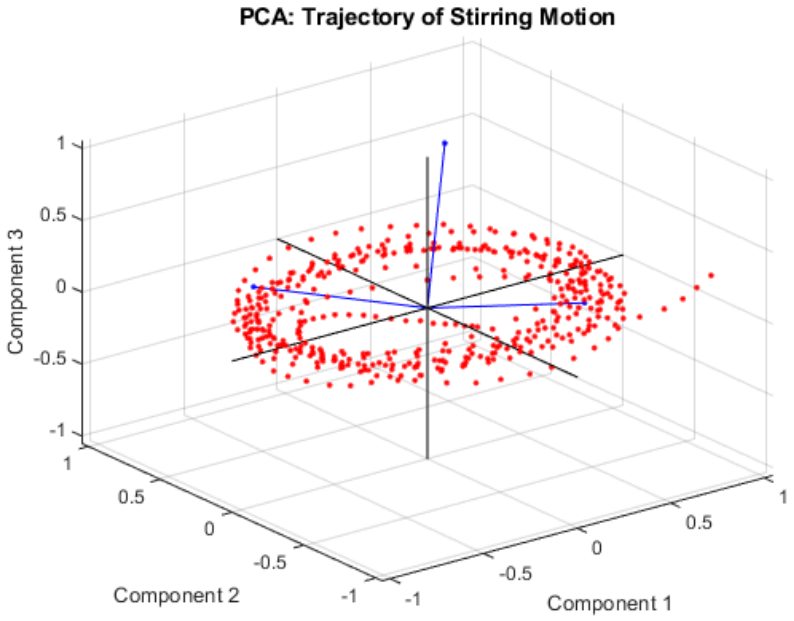


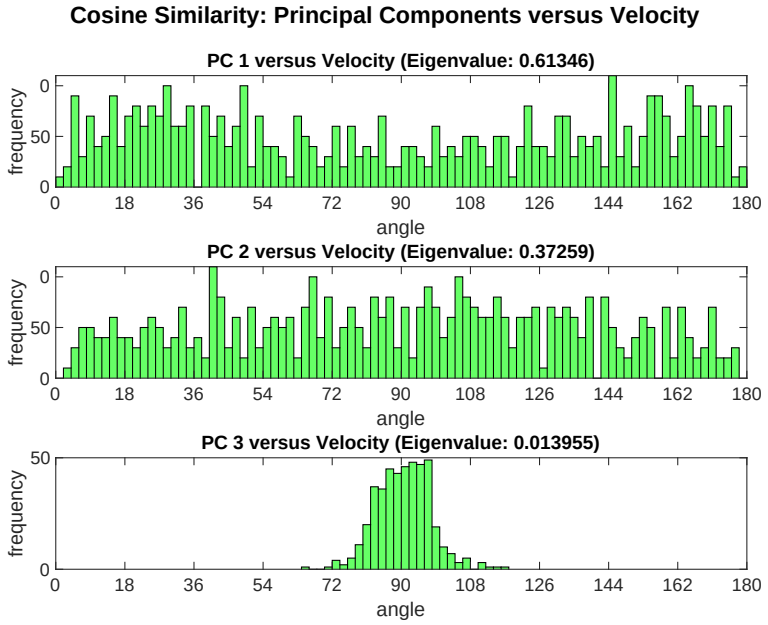
Figure 4.3. An illustration of the adapter used in the data collection in [1] (best viewed in colour). The Patriot sensor (in blue) measures position and orientation, while the ATI Mini40 sensor (in green) measures force and torque. They are aligned to the world frame (in purple) for analysis.

said manipulations. Using provided position and orientation data, which is available in data sets such as DIM [1] we can ascertain the trajectory type for several motions in which there is an active tool or object being manipulated.

To determine the prismatic trajectory type, we can use methods such as principal components analysis (PCA) to find the number of axes (which would be transformed into principal components, or PCs) that capture(s) the most variance of the trajectory. We considered that the number of DOF for a motion is reflected by the number of PCs that would capture about 90% of variance. Motions such as flipping with a turner are effectively 1D (and in minor cases 2D) motions because a single PC captures about 90% of the variance of those trials. Mixing, beating and stirring (which are all variations of the same motion) data confirm that the motion is 2D since the combination of the 1st and 2nd PCs met our requirements; this can be observed in the projection shown as Figure 4.4. One can compare the derived PCs to the velocity (i.e. directional vectors between trajectory frames) to also support whether motions exist within those dimensions using cosine similarity. Should the velocity vectors align with the PCs, we would expect values closer to 0° or 180°. In Figure 4.4b, not only does the 3rd PC contribute very little in capturing the motion, but it is normal to velocity (since the histogram shows a prevalence of vectors with cosine similarity peaking around 90°).

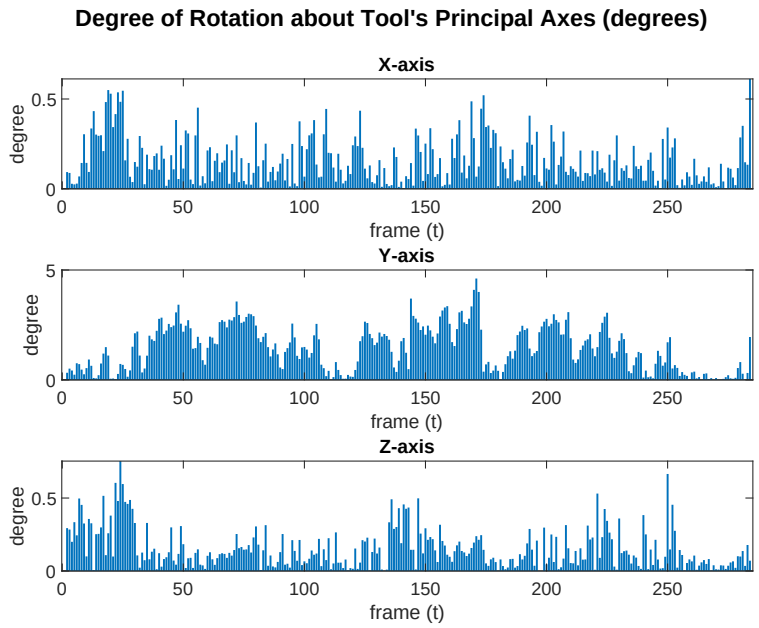


(a) Projection of trajectory via PCA (stirring)

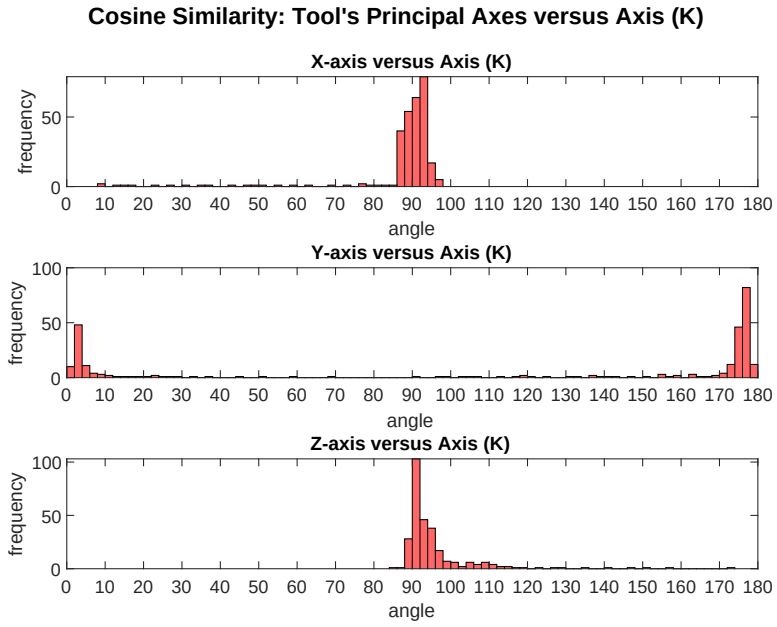


(b) Cosine similarity: linear velocity versus PCs (stirring)

Figure 4.4. Example of how PCA can be applied to recorded position data to derive prismatic bits of motion code for the 'stir' motion. In Figure 4.4a, the trajectory's points lie on a plane, hence it suggests that this is a 2D prismatic motion. In Figure 4.4b, which shows a histogram of the number of velocity vectors and their similarity to each PC, it is further supported that the motion primarily lies in PCs 1 and 2 (capturing ~99% of variance). It can also be observed from the projection that this trajectory shows recurrence since the motion is cyclical.



(a) Degree of rotation about principal axes (loosen screw)



(b) Cosine similarity: axis K vs. tool's principal axes

Figure 4.5. Example of how the axis-angle representation can be used to identify revolute bits of motion codes for the 'loosen screw' motion. In Figure 4.5a, we illustrate the change in rotation about each axis with respect to the last frame's orientation – suggesting significant rotation in the y-axis –, and in Figure 4.5b, we compare each frame's axis K to the tool's principal axes. Figure 4.5b suggests rotation about the y-axis, hence making it a 1D revolute motion.

To determine the revolute trajectory type, we can convert the position and orientation data to rotation matrices and measure the amount of rotation about the principal axis of the active tool. The axis-angle representation (which represents a frame as a vector K and an angle of rotation θ) derived from rotation matrices can also be used to compute the angle of rotation based on θ. A significant rotation about this principal axis suggests that there is at least one axis of rotation. In Figure 4.5, we illustrate how we can extract revolute properties for the motion of loosening a screw. Given that the tool's principal axes are defined as in Figure 4.3, we expect that the operation of a screwdriver requires major rotation about the y-axis. This is supported by Figures 4.5a and 4.5b.

4.4.2 Experiment: Observing Similar Motions from Demonstration Data

In this section, we discuss our experiments using DIM to determine whether we can determine similarity between motions based on demonstration data. In other words, we want to see whether the motion clusters formed by translating motions to code (or if motions are paired close together even if they do not share the same code) are corroborated by the data. However, due to the limitation of the force sensor in the data collection process for DIM, this data set does not have manipulations involving high force or torque, such as squeezing, mashing, or pressing. Furthermore, we did not analyze non-contact motions (such as pouring or sprinkling/shaking) because there are no interactive forces to measure between active and passive objects. It is for that reason we do not have mappings to all motion clusters. Several motions were collected as multiple variations of demonstrations, and so we try to combine all recordings in this data set. These experiments were conducted and presented formally in [114].

4.4.2.1 Methodology

Using the force data from DIM, we created a representative model for each motion type using Gaussian Mixture Models (GMM). Each GMM represents a force distribution across space to derive a motion description of a motion type, and they are built by combining the data points generated in multiple trials of demonstrations. To measure the similarity of motions using their individual force distributions, we use the Kullback-Leibler (KL) divergence method [124]. The typical method for

measuring KL divergence between two distributions is to use random sampling between different points; however, this is a very intensive task for us to do with GMMs, and so we used the variational approximation of KL divergence (as proposed in [125]) as the distance measure between a pair of different motions. Originally, this metric is asymmetric and it is non-transitive (i.e. the KL divergence value from A to B will not be the same as that from B to A). However, we can obtain a symmetric result by taking the average of the divergence values obtained from the two sets of pairs (i.e. we take the value from A to B and B to A and computing the average). Since we have multiple recordings for certain motion types, we also computed the average of all KL divergence values computed for each of those instances. This makes it easier to interpret the pairwise values we obtain, which we present in a matrix form as Figure 4.6. The values obtained from KL divergence are unbounded and non-negative, where the closer the value is to 0 (based on colour, the deeper the shade of the blue), the more two distributions are considered to be alike; conversely, the larger the value obtained from this calculation (based on colour, the lighter the shade of yellow), the more dissimilar two manipulation motion types are from one another based on force readings. Matrix values are symmetric, so we omitted the upper diagonal values.

4.4.2.2 Discussion

The main question we will be addressing in this section is: how well do our motion clusters match real supporting data? We determine this by looking at how similar motions classified as certain clusters match up to others that are also considered to be in the same cluster based on force/torque readings. In Figure 4.6, we have certain activity pairs whose motion labels agree with our taxonomy such as: mashing to slicing, mashing to shaving, spreading to shaving, spreading to mashing, peeling to shaving, and twisting for both directions. There are several motions which are close to one another but differ to the clusters in Table 4.1 due to one or two attributes. Even though brushing and shaving are considered different in the taxonomy, this is only due to the nature of the tools; brushing is considered to be manipulator-deforming, while shaving is manipulatee-deforming. The movement type and force application are expected to be similar aside from the deformation type found in these tools, and therefore these motions can be considered to be similar.

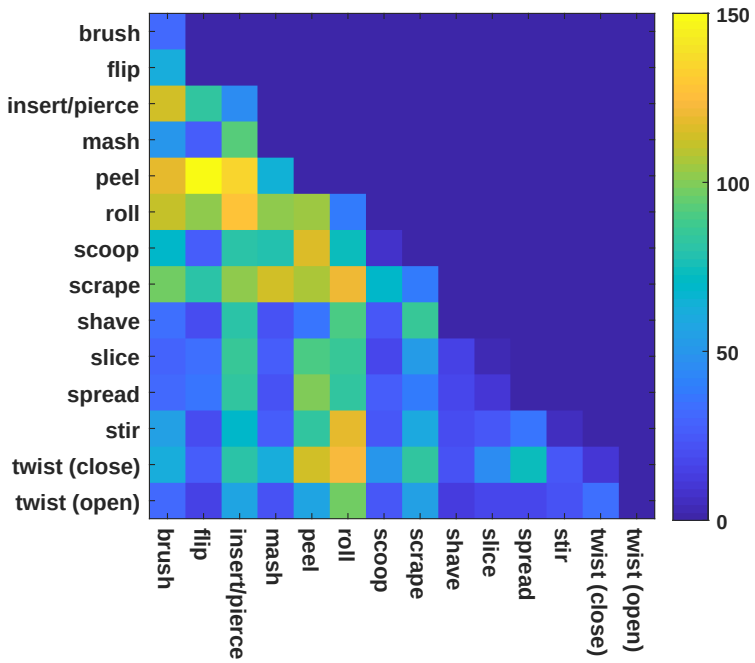


Figure 4.6. Matrix showing the Kullback-Leibler divergence values computed using DIM (originally from [114]). We only show the lower diagonal since the computed matrix is symmetric (this image is best viewed in colour).

Similarly, flipping and scooping are similar to one another because they are both prismatic and revolute; however, flipping is considered as a rigid engagement motion, while scooping is an admitting, soft engagement motion. Inserting/piercing is considered to be somewhat distant to all other motions, with perhaps the closest to twisting, which does not match our expectations.

Other pairs which we expected to be similar but they did not have low KL divergence values include peeling and scraping; conversely, motion pairs that were deemed similar but do not match our taxonomy include flipping and mashing, flipping and shaving, stirring to slicing, stirring to shaving, and stirring to spreading. Twisting open is found to be similar to many other motions such as slicing and shaving which are not revolute but prismatic only motions. This illustrates that these features should not be neglected when comparing motion data. Since the KL divergence only considers force readings, we neglect other factors which may give away unlikely matching candidates, which are likely to be obtained from an analysis of motion trajectory data or video analysis. This is why some similarities do not match with the intra-clustering of motions.

4.4.3 Experiment: Comparing Motion Codes to Word2Vec

To show how motion codes produce more ideal results in measuring distances between motion types, we show how motion vectors from Word2Vec, which are derived from natural language, is not sufficient to represent manipulations in classification algorithms. We contrast this representation to another popular word embedding technique, Word2Vec [101], which creates a vectorized representation of words directly from natural language, to show that it is not suitable for determining similarity between motions. Word2Vec is a very popular method for representing words or text as multi-dimensional vectors for use in natural language processing tasks with neural networks. Typically, each word is initialized as random vectors, whose distances are continuously adjusted with respect to other word vectors. Words are related based on locality; that is to say, if one word is frequently seen among neighbours of that word in source text, then its vector along with its neighbouring words' vectors will be closer to one another than other words in the vocabulary.

4.4.3.1 Methodology

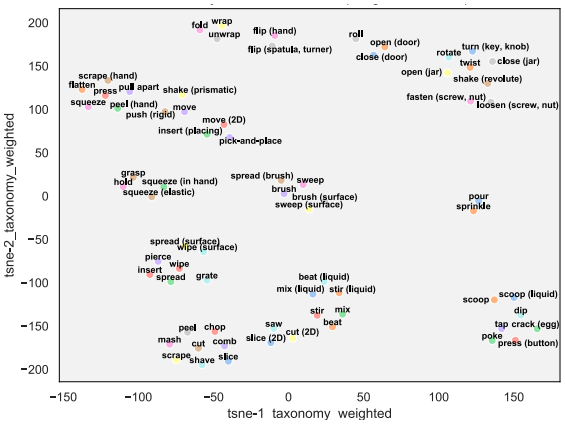
To compare both motion codes and Word2Vec embeddings, we used dimension reduction with PCA and then t-SNE [126] to visualize these word embeddings in 2D and to observe relative distances between each word vector. With t-SNE, we used a perplexity value of 15; by using a reasonable perplexity value, we control the extent to which motion vectors cluster with one another. Although certain motions will be assigned the same code, the t-SNE algorithm will position their projected vectors in close yet non-overlapping positions; similar motions would be clustered near each other since t-SNE preserves local neighbours while keeping dissimilar motions far from each other. In Figures 4.7a, and 4.7b and 4.7c, we see the relationship between motions based on motion codes, while in Figures 4.7d, 4.7e and 4.7f, we see the 2-dimensional projection of motions based on pre-trained Word2Vec models from Concept-Net [127], Google News and Wikipedia [128]. Distances in t-SNE for Word2Vec vectors were measured using the cosine similarity metric; with motion codes, we used the regular Hamming metric (Figure 4.7c and a weighted variation that we defined ourselves. Using a weighted approach allows us to emphasize dissimilarity based on key motion taxonomy attributes rather than the regular Hamming metric, which measures the

degree of dissimilarity among bits with no considerations for their meanings. We illustrate the difference between two variations of distances for t-SNE as Figures 4.7a and 4.7a respectively. In Figure 4.7a, a higher weight is assigned when two motion code vectors are different in interaction type (contact), while Figure 4.7b places more emphasis on motion trajectory type. We defined two weighted values α and β, where α is set to 5 and β is set to 2. α was used as the penalty when two motions are of different interaction type (i.e. contact versus non-contact), reflected by the 1st most significant bit (MSB) or if two motions are of different trajectory types (7th to 10th MSB). β is used as a secondary penalty among similar contact interaction codes with differing sub-attributes (2nd to 5th MSB); it wass also used as penalty for recurrence (6th MSB) and manual operation (11th MSB). All other combinations were measured normally with a penalty of 1.

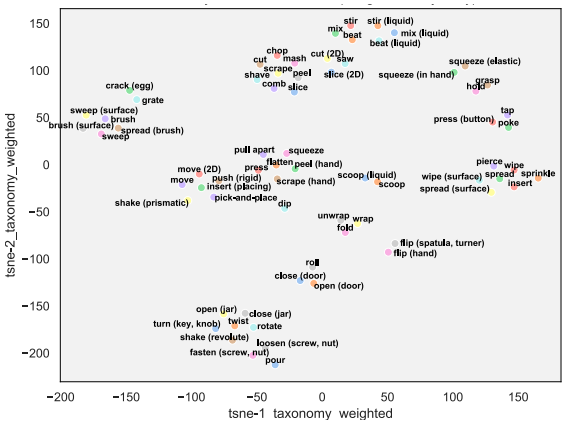
Word vectors are associated with single words, so vectors of functional variants of labels that we have listed in Table 4.1 cannot be found directly. For instance, the labels *'mix'* and *'mix (liquid)'* are different based on the permanence of deformation. Some motions were substituted with other words, such as *'pick-and-place'* to *'move'*.

4.4.3.2 Discussion

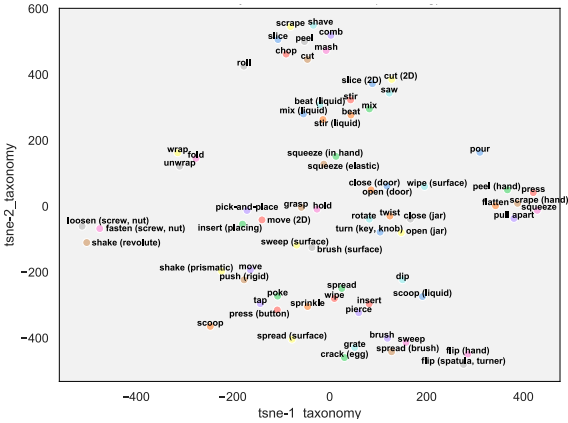
As seen in the t-SNE plots presented as Figure 4.7, using motion codes for embedding will result in the placement of functionally similar motions (i.e. those that are close in Table 4.1) close to one another in a different way to Word2Vec embeddings. Motion codes end up naturally clustering motions of similar attributes while distancing those that are functionally different as other clusters. Using a weighted approach rather than the regular Hamming distance between motion codes preserves neighbours better. The major disadvantage of using Word2Vec vectors is that we are unable to represent or capture multiple senses or meanings for a single word label. Furthermore, there is no way of discerning between different forms of a word such as parts of speech. For instance, in Figures 4.7e to 4.7f, *'pour'* is placed closest to the word *'tap'*; since the word *'tap'* in the English language can either be a verb or noun, the word is perhaps understood in the context of the noun. This would probably make more sense since water usually flows or pours out of the tap. However, when considering the manipulation in a mechanical sense, it does



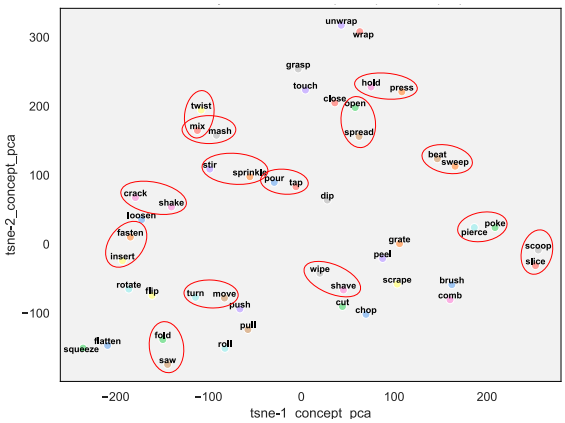
(a) Motion Codes (Contact)



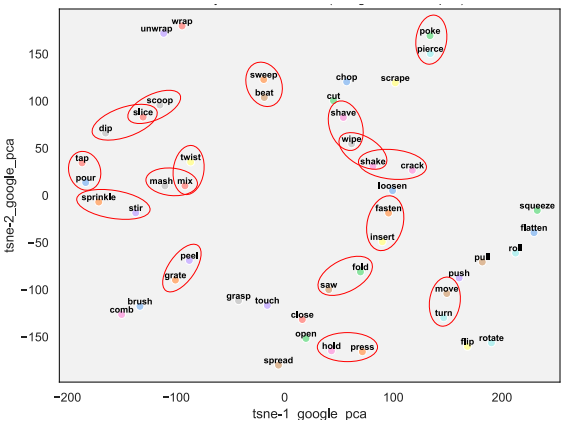
(b) Motion Codes (Trajectory)



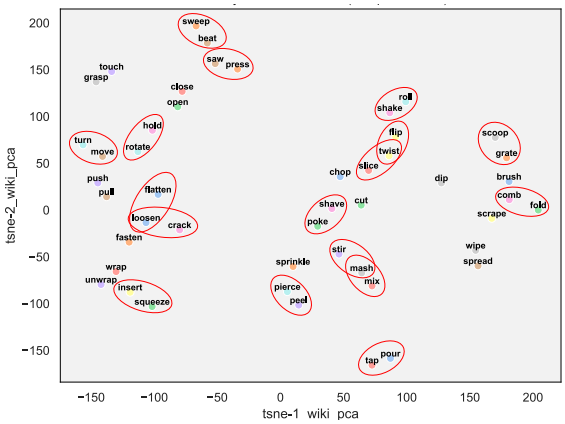
(c) Motion Codes (Regular Hamming)



(d) Concept-Net



(e) Google News



(f) Wikipedia 2018

Figure 4.7. Graphs showing the 2D projection of vectors as a result of t-SNE from: a) motion codes with higher weight on contact features, b) motion codes with higher weight on trajectory features, c) motion codes with regular Hamming distance, and Word2Vec embeddings from d) Concept-Net, e) Google News, and f) Wikipedia 2018. We highlight certain examples of motions that do not share mechanical equivalences in d) - f) with red circles (best viewed in colour).

not match our expectation since their functional attributes are different, where *'tap'* is considered as contact and prismatic and *'pour'* is non-contact and revolute. Instead, using motion codes, if we prioritize trajectory type (Figure 4.7b, the label *'pour'* is placed to other revolute-only motions such as *'twist'*, and *'fasten (screw, nut)'* (although being a cyclical motion); if we prioritize contact interaction type (Figure 4.7a, the label *'pour'* was placed closest to the label *'sprinkle'* since it is also non-contact while being placed further away from contact engagement motions. Other Word2Vec results that do not match functionality (which we highlight with red ellipses) include *'beat'* and *'sweep'* (Figures 4.7d and 4.7f), *'stir'* and *'sprinkle'*, and *'mash'* and *'mix'*. Other than the highlighted motion pairs, Word2Vec embedding generally captured the connection between certain labels such as *'cut'*, *'slice'*, and *'chop'* since these are synonymous to one another.

Another shortcoming of Word2Vec embeddings is that we are unable to effectively compare functional variants of motion types, which was the reason behind us simplifying labels to single words. However, this leads to ambiguity in motion labels since we cannot be very descriptive using one word. For example, the motion labels *'open door'* and *'open jar'* were simplified to a single word vector for the label *'open'*, but the sense of opening can differ depending on the object we are manipulating. With the two separations *'open door'* and *'open jar'*, although they serve a similar purpose, the way the motion is executed is different (mainly prismatic versus revolute trajectories), and these mechanics should be considered in evaluating differences between motions. Other labels that also resulted in this include *'squeeze'* (both elastic and rigid), *'mix'* (liquid and non-liquid) and *'press'* (unimanual and bimanual). With motion codes, since we only view motions from the mechanical perspective of robotic manipulation, we avoid issues in ambiguity from using natural language. Suggesting labels can be as easy as identifying specific traits and identifying a label that is closest to existing instances. The representation of manipulations in an attribute space can be likened to the idea behind *zero-shot learning* (ZSL); just as in ZSL, even if we do not know the human labels for certain class instances, motion codes capture attributes that intrinsically represents such classes. Motion codes can also facilitate learning unknown manipulations if we already know how to execute other manipulations with similar codes.

4.5 Future Work and Ideas Yet Explored

The motion taxonomy was proposed for the consolidation of motion aliases to thus resolve ambiguity from using natural language labels and for representing motions with respect to attributes or characteristics of motions. To derive motion codes for manipulations seen in demonstration videos, we would need to develop specialized classifiers (such as neural networks) to identify subsets of characteristics and output substrings, which can then be concatenated together to build the entire code. The use of motion codes is akin to zero-shot learning approaches, where existing labels can be used as reference to label unseen or unknown instances. However, this itself is an ongoing research problem, where we aim to evaluate the performance of taxonomical motion codes in video understanding and activity recognition tasks when compared to state-of-the-art methods that may or may not consider label embeddings when training these models. Additionally, once we have investigated the usefulness of this taxonomy, we will incorporate motion codes as labels in our FOON representation.

4.5.1 Considering Other Features for Motion Codes

Presently, our group is investigating how efficient motion codes are for the purpose of motion classification. The challenge there first lies in the development of an effective way to generate motion codes directly from videos of manipulation demonstrations, such as cooking or assembly videos. The motion taxonomy is not limited to the features proposed in this chapter, since there may be certain features that could not be useful depending on the context or situation or there may be no way of acquiring select features. For instance, motion trajectory attributes may be challenging to get – especially in the case of videos that are recorded in 2D. We may not have this problem if we have tools that can be used to record the trajectory of demonstrators. Additionally, we may consider other important manipulation cues or indicators such as grasp types, which can be obtained from using grasp taxonomies as introduced in the beginning of this chapter. We could possibly explore the area of contact mechanics to further define or describe contact between active and passive objects. In addition, we could identify classes of tools or objects as well which can further divide certain motion code classes that may need to be segregated.

4.5.2 Using Motion Codes in FOON

As stated before, one primary purpose of the taxonomy is to translate motion labels into a common language – a machine-level language understood by robots – for the goal of creating functional units to expand FOON from other sources of manipulation data. Motion codes can be used in place of motion node labels as we have manually defined in FOON; we can simply generate motion codes from the videos obtained in data sets or on the internet. This would greatly reduce the complexity of annotation, as there are many state-of-the-art classifiers for object recognition that can be coupled with a motion code generator to create FOON subgraphs. We can also use annotated data from other sources such as EPIC-KITCHENS [109] and MPII Cooking Activities Dataset [113] and avoid trying to translate their language to our FOON language and labels.

4.5.3 Motion Generation from Motion Codes as Blueprint

Another important question that needs to be addressed in the future is how motion codes can be used for motion generation. We have yet to determine the extent to which they apply to motion generation beyond simply describing how the manipulation should be carried out or executed by a robot replicating the manipulation, as a motion planner would require less abstraction to determine how a motion should be done. In the current state of motion codes, we can only use it as meta-data, since it would indicate what we should expect when a manipulation is carried out on certain objects, which would also be indicative of the type of material they are made of. Using demonstration data like DIM allows us to extract other details such as range of motion for each dimension, which could be a very important hint to generating motions.

Chapter 5: Task Planning through Knowledge Retrieval

One of the main purposes of FOON is to equip robots with the knowledge they need to solve manipulation problems in the household. Knowledge is retrieved from FOON through the process of *task tree retrieval*, where a *task tree* is a subgraph with the series of steps it needs to execute to solve the problem. The task tree has the potential to be entirely novel since a FOON will comprise of knowledge spanning many demonstrations; in other words, a robot can draw from multiple sources of knowledge to execute an entirely unique sequence of actions for a given activity. Given a desired goal (presented to the robot) and a set of available objects in the kitchen, formally, there are two steps in generating and executing manipulations from the FOON: 1) retrieving a task tree from the universal FOON, and 2) generating the motions needed to accomplish the task. This algorithm draws ideas from combination of the breadth-first search (BFS) and depth-first search (DFS) algorithms as a specialized application of the branch-and-bound algorithm.

5.1 Introduction to Task Tree Retrieval

As input to the algorithm, a goal node N_{Goal} is identified and given to the robot as a desired target or product; this node must exist in the universal FOON. Additionally, the robot needs to know about what objects exist in its environment, which is presented as a list K. Initially, the task tree T will be empty; through task tree retrieval, T will be populated with a series of functional units that ultimately results in N_{Goal}. The searching algorithm is driven by three important structures: the list of kitchen items (denoted as K), a list of items we do not know how to make (denoted as S),and a list of candidate functional units that produce nodes removed from S (denoted as C). To preserve the order in which we explore nodes, it is best to use a queue data structure for S. We

This chapter was partially published in [2]. Permission is included in Appendix B.

begin by adding N_{Goal} to S since we initially do not know how (and if) the robot can make that object. We then dequeue the head of the queue, which we denote as H, and search the universal FOON G_{FOON} for all procedures that create it (i.e. functional units whose output object nodes contain H); we would add these units to C. Once we have identified all possible candidate units, we then proceed to selecting the ideal unit to add to our final task tree T. The ideal candidate unit $FU_{candidate}$ is that which we can execute in its entirety, meaning that we have all the required input object nodes in K. If all input objects in N_{Input} for a given unit are present in K, we can add this unit to T and mark H as seen and "solveable". However, if there are no functional units that can be executed fully due to missing objects, then for all candidate units, we add those items from N_{Input} to the queue S so that we can determine how we can make them. These objects added to S can be considered as subgoals that need to be solved in order to solve the main goal.

At this point, we move to the next iteration of the algorithm by removing the proceeding head in S and then repeat the search for candidate units that contain H in N_{Output}. The search continues as long as there are items remaining in the queue S – in particular, N_{Goal}. We will know that a task tree sequence is found when N_{Goal} has been marked as being "solveable", i.e. when N_{Goal} eventually gets added to K. If there are no possible solutions for N_{Goal} using the present K, then we would keep encountering the same objects over and over again and the queue's size remains constant. In this case, the search would have to be repeated with the missing objects needed to solve the problem, or the robot would need to be taught alternative ways of solving the problem.

5.1.1 Analysis on Task Tree Retrieval Algorithm

Given that the algorithm depends on an input set of objects available to the robot in its environment (i.e. the list of kitchen items K), this allows us to find a solution that suits the current setting. If we were to simply consider the number of possible unique paths (which is what we explore in Chapter 7), this would significantly worsen the time complexity, and so this trade-off must be made to find a solution in real-time. The task tree retrieval algorithm is also akin to the reachability problem in Petri Nets. These problems themselves are polynomial in complexity. However, by including the list of kitchen items K, we reduce the complexity of this solution as a greedy

Algorithm 2: Task tree retrieval from universal FOON (greedy approach)

{*As input, we provide the goal node and list of initial kitchen items:* }
Let N_{Goal} be the target goal node and K be list of objects found in kitchen
Let S be queue of objects to search, T be final task tree, H be head of S, C be list of candidate functional units whose output nodes contain H
Check if node N_{Goal} exists in G_{FOON}
Add N_{Goal} to S

{*The search iterates until the goal node is present in the kitchen:* }
while N_{Goal} not in K **do**
  Dequeue H from S
  {*We search for all functional units whose outputs contain head of queue:* }
  for all functional units FU_i in G_{FOON} **do**
    if H in N_{Output} of FU_i **then**
      {*Add to list of candidate units:* }
      Add FU_i to candidates list C
    end if
  end for

  {*Find ideal candidate unit (whose input objects are ALL in kitchen, if possible):* }
  for all functional units $FU_{candidate}$ in C **do**
    for all nodes N in N_{Input} in FU_i **do**
      Let $count = 0$
      if N not in K **then**
        {*Items not found in kitchen must be added as subgoals to the queue:* }
        Add N to queue S
      else
        {*Variable 'count' is used to tally the number of input objects that are present in the kitchen:* }
        $count$ += 1
      end if
    end for

    {*If we found all input objects in the kitchen (i.e. 'count' is equal to the number of input nodes), then we add functional unit to the final tree:* }
    if $count == |N_{Input}|$ **then**
      Add $FU_{candidate}$ to T
      $C = \emptyset$
    end if
  end for
end while

return T if $T \neq \emptyset$

algorithm, since we will favour functional units that immediately meet the requirements we have. Therefore, if we consider that a solution can be found, then the complexity would be $O(|E| \cdot |P|)$, where $|E|$ is the number of edges (or functional units in this case) based on the number of units in a path P (bounded by the maximum length of a path or diameter of the graph). However, if a solution does not exist, this means that there are certain inputs that are needed but missing from the environment. To prevent this algorithm from executing infinitely, we terminate the search if the queue of items to search S does not increase nor decrease in size for a certain number of iterations of the algorithm.

5.1.2 Example of Task Tree Retrieval

We illustrate an example of task tree retrieval using the example FOON shown in Figure 5.1. We use the same merged subgraph from Chapter 2 to obtain a task tree, which is shown as Figure 5.2. Here, the objective is to determine how to cut an orange into divided halves (highlighted in purple) given a set of objects observed in the scene (highlighted in blue). With respect to Algorithm 2, the purple node would be N_{Goal} and the blue nodes would be added to K. Upon the first iteration of the searching algorithm, we can observe that there are three inputs needed to output the goal node. However, we only have one of those items in our environment, i.e. the *knife* object. The other two nodes are then added to the queue S along with the original goal node since we did not satisfy its requirements as yet; these subgoal nodes are shown in orange. However, upon the next few iterations, we then see that the required objects are in the kitchen, and so we derive the task tree in Figure 5.2. When executing the final task tree, the robot can begin task execution by manipulating these blue nodes to incrementally work towards the goal node; in other words, these nodes can be viewed as root nodes as found in basic tree structures, except that trees typically do not have multiple roots. The path we take is entirely dependent on the availability of the objects in the robot's environment.

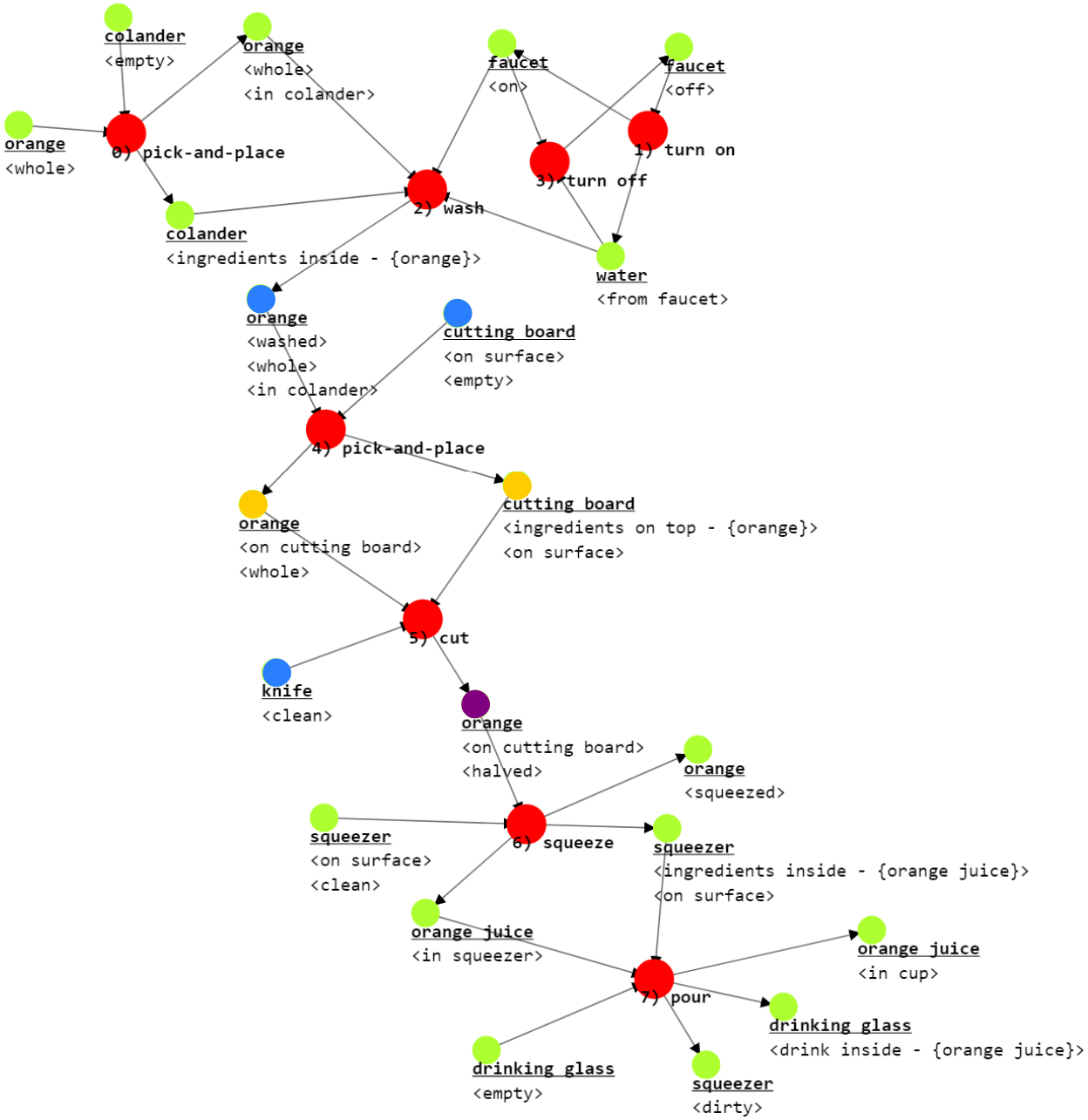


Figure 5.1. Using this subgraph describing the preparation of orange juice, a robot can derive the knowledge of cutting oranges (node in purple) using available objects in its surroundings (nodes in blue).

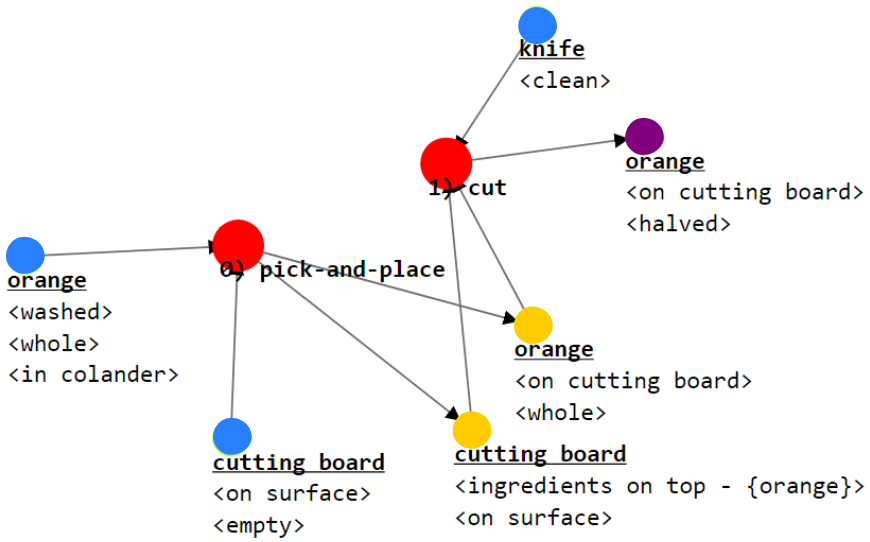


Figure 5.2. Task tree showing the steps needed to prepare halved oranges (highlighted in purple) using available objects (nodes in blue) using knowledge from Figure 5.1's subgraph.

5.1.3 Novelty of Task Tree Retrieval

The ability to merge and combine knowledge into a single universal FOON makes our network very powerful and useful for solving a wide array of problems. Within a universal FOON lies many possible task trees for different scenarios and variations of tools or ingredients. We can uncover entirely novel ways of executing tasks, as there may be several ways of creating a particular meal or preparing an intermediary item. Our task sequences therefore are not necessary to follow an entire procedure from a single video source. For example, there are many ways to prepare a sauce for meat, and by using the knowledge on how to prepare sauces with a variety of ingredients, we can compensate for the unavailability of certain items needed if we instead followed one recipe. Similarly, we can prepare meals using other items as substitutes for main ingredients that are used in the conventional recipes; this idea is further explored in Chapter 6. The novelty not only comes from the possibility of different task sequences but also in the flexibility in how we prepare meals.

5.1.4 Modifying the Task Tree Retrieval

The task tree retrieval algorithm as it is presented can be considered as a greedy approach, since we select the first functional unit whose requirements we meet for each node H. However, this searching procedure can also be adjusted to make use of weights, which act as heuristics and constraints on the creation of a task tree, to result in an optimal search. These heuristics can be a *cost* value that is associated with each motion node, influencing the selection of functional units which are added to T. In Chapter 7, we explore the addition of weights that reflect the difficulty in a robot performing a specific task, which is reflected by its corresponding motion node. We discuss further modifications that can be made to the algorithm in Section 5.2.

5.1.5 Motion Generation

Once we have found a suitable task tree, the robot can then use it to generate a task sequence that contains a series of motions, objects, and their states, which provides step-by-step commands executable by a robot. After a functional unit in the task tree is provided and the involved objects are identified in an environment, a new trajectory of the motion needs to be generated using the locations of those objects as constraints. A new trajectory can be generated using techniques such as motion harmonics [129]. More recently, methods such as recurrent neural networks (RNN) can perform exceptionally well for motion generation; for instance, in [130], Huang et al. developed a model that pours liquid with very minimal error when compared to other research works. Interested readers are recommended to refer to the publications by my colleague Yongqiang Huang for related work on motion generation.

5.2 Future Work and Ideas Yet Explored

In this section, we will be addressing certain tricks or modifications that can be added to improve the run-time and performance of the task tree retrieval algorithm. Certain modifications should also be made to account for the task execution phase that follows the task tree retrieval phase (task planning).

5.2.1 Speeding Up Task Tree Retrieval

Naively, the algorithm requires iteration among all functional units in a universal FOON to identify candidate functional units to be added to the final task tree. To save some time in searching, one can use dictionaries or dictionary-like structures that can map output object nodes for each functional unit to its corresponding functional unit(s). Although this will require more space (as the time taken to build these dictionaries at run-time may be significantly long), if a robot's universal FOON does not get updated frequently, a structure like this would help to speed up the searching process. In the code described in Appendix A, these dictionaries are used and implemented for the searching procedure. We describe how these dictionaries are built when the code is executed in Appendix A.

One shortcoming to the algorithm is that in finding a solution to a specific object, we end up adding several new object nodes to the queue of items that have yet to be marked as 'solveable'. Because of this, even though we may have found a way to make a particular item, there may be existing subgoals remaining on the queue S that end up being added to the final task tree. This is common when we have a very large list of kitchen or environment objects since we may be able to satisfy a large number of requirements. We end up adding functional units that are no longer relevant or needed to be executed. To remove persisting subgoal objects whose dependant goal node has been solved, we can add an extra list structure that removes all subgoals of a node H when we have added H to the list of kitchen items K.

5.2.2 References to Source Recipe

As mentioned in Chapter 2, FOON presently lacks information about portions or quantities, which would be necessary when accounting for serving sizes or party sizes (of people). After deriving a task tree, there should be some way of referencing the original recipe from which a functional unit was obtained in order to get an idea of the ratio between the ingredients for a typical serving of a meal. This may be easier to handle when dealing with higher levels of hierarchy, as it would be more unlikely to deal with overlapping units.

5.2.3 Handling Unseen Cases of Items

There is a duality that exists with a universal FOON: although we have the power of representing several demonstrations as a single large network, we are still limited to what is found in the universal FOON. A robot will be limited to the knowledge that has been annotated and a robot can potentially fail to perform tasks because of a lack of flexibility or novelty in its task trees. A human would be able to improvise by applying what the person knows to still find a solution to a problem. This intuition of improvisation leads us to the developments of the following chapter. However, without this development, it would require researchers or developers to ensure that a universal FOON is as complete as possible, where there are no gaps of information.

Chapter 6: Generalizing Knowledge in FOON

As discussed in the prior chapter, the power of a universal FOON lies in the novelty of task tree solutions that can be found using task tree retrieval. However, with our present methodology, learning new actions or tasks as FOON graphs requires annotating additional video demonstrations by hand and merging their subgraphs with the universal FOON, which for us is a very time-intensive process. Hence, the solutions we can acquire from FOON in task planning is limited to what we have in FOON. To create graphs automatically is inherently a difficult problem due to the major challenge in recognizing object states and motions in 2D videos from the Internet without additional modalities of information. Although we have shown how FOON can be used for video understanding in [102], it is still a challenge to acquire accurate and complete annotations.

In prior work [103], we investigated a means of creating useful knowledge in FOON without having to annotate new videos. We introduced two methods of generalization of knowledge contained in FOON: *expansion* and *compression*. These two methods use the idea of object similarity to decide upon what knowledge can be extended to other objects, especially those not present in FOON. The intuition behind object similarity is as follows: if we know how to manipulate a certain set of objects, then we can also manipulate those which are similar to it. From another point-of-view, if we do not have a specific object, we can use items which are similar to it to complete the task. Therefore, if we were unable to find a solution using the task tree retrieval algorithm of Chapter 5, which is due to missing information pertinent to the current scenario, we could possibly find other objects that can substitute those that have not been found in the environment.

In this chapter, we shall discuss the details of each method as it pertains to the generalization of knowledge in FOON and discuss results of knowledge retrieval using such approaches.

This chapter was partially published in [103]. Permission is included in Appendix B.

6.1 The Concept of Object Similarity

The idea behind the proposed generalization methods is based upon the concept of *semantic similarity*. We have explored the use of semantic similarity to measure relatedness between objects we know how to manipulate in FOON and those for which we are missing knowledge about in FOON. This method can be beneficial for filling in gaps of knowledge. For instance, one issue we have observed in our initial version of FOON is that we will miss out on very basic steps (or functional units) because of missing information in source videos. For example, in a given activity, we may observe onions in the *"chopped"* state in a bowl, but we never see how we can obtain this specific state. As we know, onions are naturally not found in this state, but we may expect that we can obtain *"chopped"* onion by executing the *"chop"* action with a knife object on a *"whole"* and *"peeled"* onion. If a robot only detects *"whole"* onions in its environment, then it will be unable to complete a task that requires *"chopped"* onions. However, if a robot using FOON knows how to chop something *similar* in function or make to onions (such as turnips or scallions), then we can theoretically perform the same manipulation with onions.

6.2 FOON Expansion

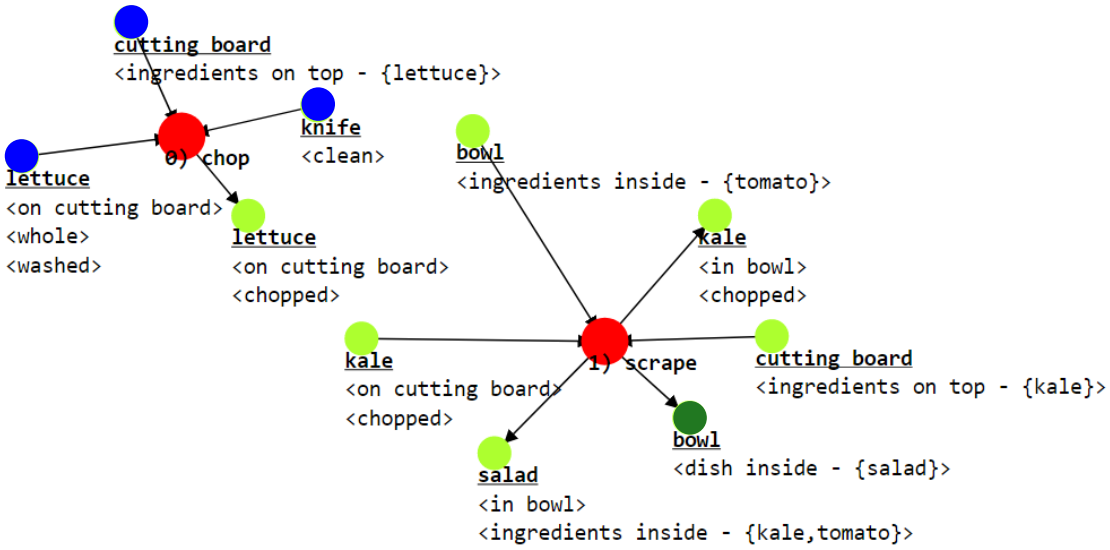
Using the intuition of object similarity, we can create new knowledge (i.e. creating new functional units) by using what we have collected as reference for new objects. The *expansion* algorithm for creating a larger FOON involves copying functional units which already exist and creating new units with similar objects. An example of how expansion can be used is shown as Figure 6.1. This will be done for every combination of objects which are similar to one another. We refer to an expanded FOON as *FOON-EXP*.

6.2.1 Creating a FOON-EXP

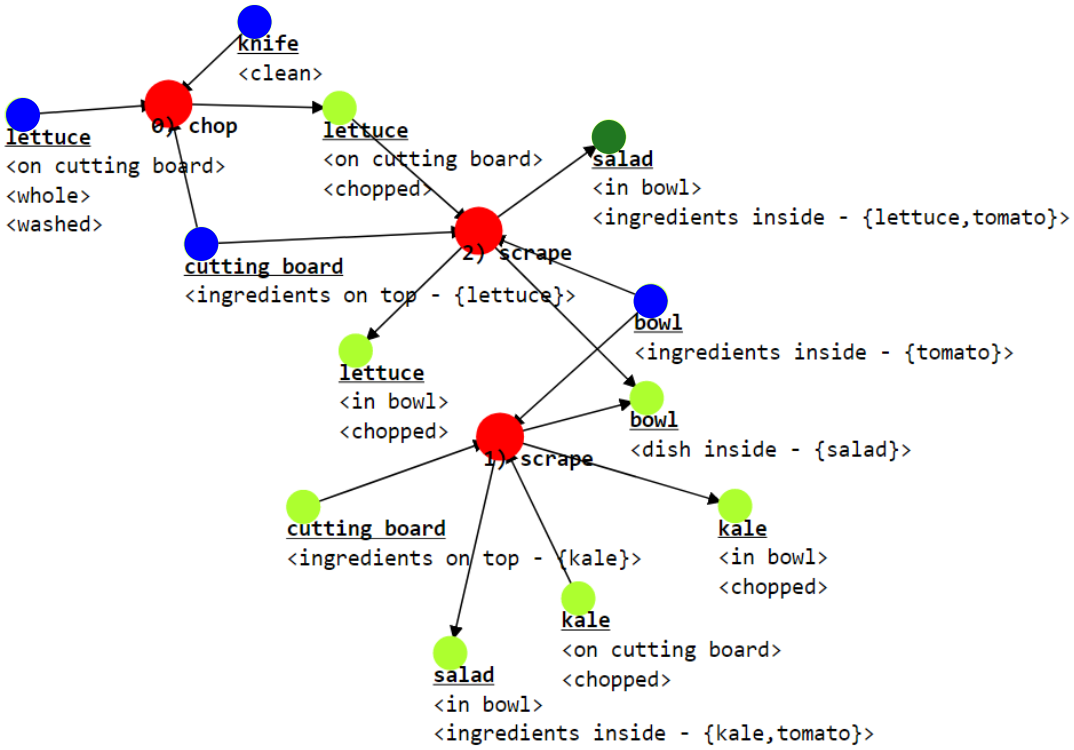
Determining object similarity requires that we can compute relatedness between a pair of objects. This either requires that we define our own criteria for similarity or we use a knowledge base from which we can measure distances between terms. In [103], we used WordNet [86],

which is a lexical knowledge base for the English language that organizes words in *synsets*, which are sets of synonyms and words that are related to one another. Using WordNet, we calculate similarity of two objects using the Wu-Palmer metric [131] available to researchers in the NLTK package [132]. This metric produces a similarity score from 0 to 1, where 1 indicates that two items are conceptually equal. Recently, we have also included Concept-Net [127], a semantic network that represents concepts as nodes and assertions or relations between such concepts, to our code for measuring similarity between words. Through a web API, we can query Concept-Net for measurement values, and we can also use word embeddings derived from Concept-Net for measuring similarity. Because the web API is limited in the amount of queries it can handle at a time, we instead use the word embedding method. In FOON expansion, we try to measure values for all object labels from either WordNet or Concept-Net. Although WordNet and Concept-Net are both remarkably impressive databases, they still have their shortcomings due to the lack of certain terms and categories that we seldomly encounter in cooking videos. There are also some objects which are not found (such as *"corn starch"*, *"muffin pan"* and *"protein powder"*), which could be alleviated manually with user-defined values. We found that Concept-Net contains more object instances, but it does not completely span all possibilities we have predefined. However, for the sake of this work, we ignore those labels that do not exist by assigning similarity values of 0 to object pairs that include any of these missing labels. To work around this, manual assignment of similarity values may be required.

When performing expansion, we would need to define a similarity threshold value as a basis for determining when two concepts are alike; in [103], we used a threshold value of 0.89 since it produced an expanded network that is easy to manage. Even a minutely lower threshold of 0.88 would result in an exponentially larger FOON, much larger than what we are using in this paper, at the cost of requiring more time and resources to complete the expansion process. In Tables 6.1 and 6.2, we outline the statistics of an expanded *FOON-100* for different threshold values to support the previous statement. For the experiments done in the remainder of this dissertation, we use two versions of *FOON-EXP* (both WordNet and Concept-Net) using a threshold of 0.9.



(a) FOON before expansion



(b) FOON after expansion (extending kale to lettuce)

Figure 6.1. An example of how expansion can fill in gaps of knowledge. Here, we wish to make a salad (node in dark green) using lettuce and other items in the environment (in blue); initially, we only have knowledge on making salads with kale (Figure 6.1a). Using object similarity, we learn that kale and lettuce are similar, as they are both leafy vegetables. We create the knowledge of chopping lettuce and adding it to a bowl with other ingredients to make a salad (Figure 6.1b).

Table 6.1. Statistics of expanded universal FOONs (*FOON-EXP*) for *FOON-100*. We show statistics based on varying threshold values using WordNet.

Threshold	*Hierarchy Level*	*# of Object Nodes*	*# of Motion Nodes*	*Total Nodes*
0.9	Level 1	473	9372	**9683**
	Level 2	3723	10691	**14319**
	Level 3	15282	16724	**31996**
0.87	Level 1	511	21589	**22100**
	Level 2	7561	25315	**32876**
	Level 3	41683	46723	**88356**
0.85	Level 1	526	28602	**28917**
	Level 2	8656	33337	**42095**
	Level 3	50562	58385	**108851**

6.3 FOON Compression

Instead of abstracting the objects using hierarchies, we can go further by abstracting objects to another hierarchy level: a *categorical* classification of objects. In this approach, we are not concerned about specific objects but instead we focus on how archetypes of objects (referred to as *object categories*) are characteristically used and manipulated in cooking. For example, fruits share a common trait in having seeds, and so we will cut most of them expecting the seed(s) to be in the centre; therefore, we can represent the act of cutting a fruit with a single functional unit as opposed to several units for each fruit type. This method can be seen as the opposite of FOON expansion, where we create new units for all objects and their similar labels. For the purpose of discussion, we will refer to the compressed version of our universal FOON as *FOON-GEN*. Through this means of generalization, we avoid the network "blowing up" from a drastic increase in size as a result of adding new functional units. However, a major requirement is predefining object categories for classification. One can use lexical databases such as WordNet or Concept-Net to assign object labels to categories providing that they do contain definitions for both object labels and categories.

6.3.1 Creating a FOON-GEN

We compressed our universal FOON such that object labels fall under one category, several categories, or possibly no categories if they are too unique an object; however, in our revised

Table 6.2. Statistics of expanded universal FOONs (*FOON-EXP*) for *FOON-100*. We show statistics based on varying threshold values using Concept-Net.

Threshold	*Hierarchy Level*	*# of Object Nodes*	*# of Motion Nodes*	*Total Nodes*
0.9	Level 1	434	2357	**2791**
	Level 2	2207	2532	**4739**
	Level 3	4728	3242	**7970**
0.85	Level 1	438	2644	**3082**
	Level 2	2357	2853	**5210**
	Level 3	5499	3997	**9496**
0.8	Level 1	442	3281	**3723**
	Level 2	2722	3597	**6319**
	Level 3	7450	5999	**13449**
0.75	Level 1	457	5155	**5463**
	Level 2	3533	5653	**9073**
	Level 3	12891	11427	**24256**

experiments, we ensure that objects can only belong to one category or no category. Initially, we defined a list of 56 categories [103]; we increased this to 57 categories, where we revised the label names and the objects that fall under each. These categories include *spices*, *open containers*, *condiments*, *vegetables* (and further classification as *leafy vegetables* or *root vegetables*), *cutlery*, and *eating utensils*. These categories have been defined in terms of functionality, but we can still account for other features such as shape and texture. We filled each category using WordNet and then we corrected them manually by allocating and grouping items ourselves. This is because WordNet lacks certain concepts as we have in our list of items, or more commonly, items were found and misclassified, even through there was a similarity of object and category found.

With the object-category index mapping all possible object labels to 57 categories, we can then construct a FOON labelled in terms of categories. The procedure to create a FOON-GEN using categories is a simple process: we iterate through all of the functional units and we search for all objects that belong to a specific category for all categories. Once we find these objects, we can simply replace them with the category and then append these new units to a separate list. Therefore, these functional units do not refer to a specific object but instead it will use the

generalized concept of an object (once an item has a mapping to the list of categories). An example of how FOON compression works is shown as Figure 6.2.

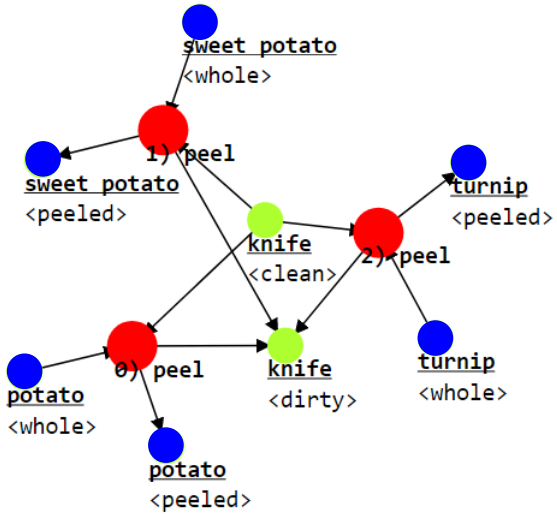
The version of FOON-GEN created from *FOON-65* is a Level 2 FOON, and it has a total of 1643 nodes comprising of 822 object-category nodes and 821 motion nodes. The version of FOON-GEN created from *FOON-100* is a Level 2 FOON, and it has a total of 1643 nodes comprising of 822 object-category nodes and 821 motion nodes. This is a smaller fraction of the number of functional units featured in our universal FOON and its FOON-EXP version. A smaller graph would allow for faster searching times so we expect that FOON-GEN would perform well in task tree retrieval.

6.4 Evaluating the Usefulness of FOON Expansion and Compression

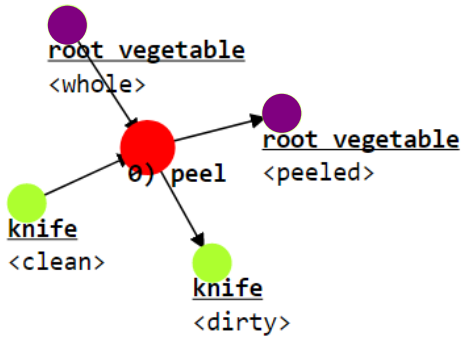
Having established and defined the concepts of expansion and compression, we justify the usefulness of our approaches and explore whether they show an improvement in solving unknown problems. In [103], using *FOON-65*, we hypothesized that our compression method, which uses categories for generalization, will outperform an expanded FOON and a regular universal FOON.

6.4.1 Methodology

Our experiments were conducted as follows: during a series of 10 trials, we randomly select 50 goal nodes as target products and we attempt to find a task tree that produces each of these goals. The network with the most successes (defined as the tally of objects for which a task tree was found out of the total number of goal nodes) will indicate the best of the three methods. These goal nodes are those which are not in its basic state, i.e. it must be the output of a functional unit. In this way we do not consider items which we may be searching for which are already in our kitchen. In each trial, we simulate different kitchen environments by randomly selecting a subset of ingredients/utensils out of a pool of kitchen items since our knowledge retrieval algorithm requires a list of items in the environment. In [103], the kitchen object pool contained 224 object nodes. We will be measuring each trial by the average time taken for task tree retrieval in addition to the number of objects successfully found. An object with no task tree found within a certain period of time (or in our case, a certain number of iterations) is considered as unsolvable and has



(a) FOON before compression



(b) FOON after compression

Figure 6.2. An example of how generalization through compression works on a subgraph. Here, we wish to make a salad (node in dark green) using lettuce and other items in the environment (in blue); initially, we only have knowledge on making salads with kale (Figure 6.2a). Using object similarity, we learn that kale and lettuce are similar, as they are both leafy vegetables. We create the knowledge of chopping lettuce and adding it to a bowl with other ingredients to make a salad (Figure 6.2b).

Table 6.3. Statistics of *FOON-EXP* used in our experiments on *FOON-65* (threshold of 0.89).

Hierarchy Level	***# of Object Nodes***	***# of Motion Nodes***
Level 1	232	3528
Level 2	1996	5493
Level 3	5306	6942

no existing solution. This is important when considering obtaining task tree sequences in real-time as needed by robots.

6.4.2 Results for *FOON-65*

In [103], using the previously described methodology, we ran the experiments while recording average running times (shown in Table 6.4) while noting the number of objects we successfully found task trees for out of a possible 50 object goal nodes (shown in Table 6.5) using *FOON-65*. For time complexity considerations, these experiments were run on a machine with an Intel Core i7-6500U processor (with speeds up to 3.1GHz) and 12 GBs of RAM. These results showed that *FOON-GEN* performs much better than the other networks, as we were able to find at least 68% of all object goal nodes all while using only a subset of kitchen ingredients and utensils. *FOON-GEN* also provided for the fastest searches on average, validating that a network of smaller size would require less time to search. Therefore, a generalized representation using compression would be ideal for solving problems in real-time. The *FOON-EXP* network also does fairly well when compared to the regular network, as in most instances it outperforms the regular network. Theoretically, *FOON-EXP* should allow for at least as many as the original *FOON-REG* network, as *FOON-REG* is a subset of the expanded network. However, the expansion dramatically increased the number of functional units in the network, resulting in much deeper searches that require more time to explore to ultimately find a task tree. In trial 9, we observed that *FOON-REG* outperformed *FOON-EXP*, as we found task trees for 6 more objects than the latter.

Table 6.4. Average running times over all trials of random searches with an unexpanded network (REG), an expanded network using object similarity (EXP), and a generalized network with categories (GEN). Underlined is the lowest average time, which was observed in task tree retrievals using *FOON-GEN*.

FOON Selected	***Average Time Over Trials (ms)***
REG	2240.2
EXP	18690.9
GEN	<u>837.7</u>

Table 6.5. Results of random-search experiment with an unexpanded network (REG), an expanded network using object similarity (EXP), and a generalized network (GEN).

FOON Selected	***Trials for Experiment***									
	#1	***#2***	***#3***	***#4***	***#5***	***#6***	***#7***	***#8***	***#9***	***#10***
REG	22	18	4	30	1	2	19	13	28	22
EXP	28	20	15	33	7	10	26	15	22	23
GEN	<u>39</u>	<u>41</u>	<u>39</u>	<u>42</u>	<u>42</u>	<u>39</u>	<u>43</u>	<u>44</u>	<u>35</u>	<u>36</u>

6.5 Limitations of *FOON-GEN* and *FOON-EXP*

The drawback to using an expanded network like *FOON-EXP* is that it does not perform significantly better than *FOON-REG*, which is primarily observed in the case with *FOON-65*. This is due to the addition of unusable functional units being created that do not accurately reflect the reality of how certain items can be used. However, by adding more functional units, we require a deeper and lengthier search (as suggested by the great difference in average search times in Table 6.4). A generalized approach like FOON-GEN alleviates this issue with fewer nodes for faster search times.

We have shown that a generalized FOON allows for more successful searches, as we alleviate the issue where expansion does not create the necessary functional units to solve a problem. However, despite this fact, certain issues remain when it comes to mapping these generalized task trees to the robot's planning of action. As we mentioned before, this representation is only symbolic at this point, and so we need to develop a means of mapping this knowledge to a manipulation planning system. It would then be a challenge to determine if a set of objects can be used to solve the problem or not. For example, can we use scissors in place of a knife if that's the only "cutter"

object available? How do we plan around those sort of situations? Such considerations would need to be addressed with a less general FOON.

6.6 Future Work and Ideas Yet Explored

Here, we will address future work associated with the techniques discussed in this chapter that rely on semantic similarity measurement.

6.6.1 Using Semantic Similarity in Real-World Scenarios

The experiments discussed in this dissertation were those from our work in [103]. We have yet to repeat the experiments with our latest version of FOON, *FOON-100*, due to time constraints and the need for further revision of object and state labels used. We plan to publish updated results in a future publication. It is also important for us to establish a proof-of-concept detailing how semantic similarity will be used by a real robot, as we need to identify how generalized task trees can be degeneralized to exact object instances found in the environment. We would also need to manage scalability issues that results from expansion; perhaps, it may be best to keep both expanded and compressed versions linked in such a way that we can preserve valid knowledge.

6.6.2 *FOON-EXP*: Handling Errors from Lexical Databases

Using semantic similarity to determine what functional units can be extended to apply to other objects not in FOON (or to add new ways of manipulating or using existing objects as functional units) can potentially allow us to learn new concepts without worrying about annotating videos. However, measuring semantic similarity can only be effective with a suitable source of information or knowledge base from which we can calculate the degree of similarity between object labels. Although WordNet [86] is a very extensive knowledge base (with concepts that span to the arguably more popular ImageNet [87]), the structure of labels do not particularly reflect functionality of objects, and as such, the metrics may unexpectedly insinuate relatedness between objects that are not similar at all. For instance, objects *honey* and *sugar* may be similar and interchangeable in cooking scenarios, but they cannot share the same states (e.g. sugar can

be granulated but honey cannot). One way of circumventing this is rather than calculating object semantic similarity between all object labels listed in a FOON's index, we can narrow our search to a subset of labels to reduce the amount of incorrect data. Nevertheless, it would still be better to use a knowledge base that is built with functionality in mind, perhaps like affordances, which requires time to define if we do so from scratch.

In addition, in previous experiments from [103], we did not identify just how many of the task trees made any sense. Unfortunately, we would still need to rely on verification done by humans to verify that they make sense or apply to the target node. As explored in [102], we can use Concept-Net [127] (or WordNet and any other lexical knowledge base) to suggest what states are valid for similar object pairs, i.e. to identify which state labels are semantically relevant to objects. We also need to decide on how objects are correctly degeneralized (or, in other words, taking each object category and assigning it to a specific object that is in the environment and that falls under said categories) to kitchen object instances in the task execution phase.

6.6.3 *FOON-GEN*: Further Exploring FOON Compression

The compression of a universal FOON as *FOON-GEN* proved to generate more task trees in our experiments. However, in order to ensure correct or valid task trees, appropriate object categories need to be defined. However, fine-grained categories can be challenging to define; further investigation will be made to consider categorizations that are used in existing lexical databases or in existing work on affordance. Along with a revision of FOON states, object categories can better reflect objects that could possibly share state types.

Chapter 7: Leveraging Robot's Capabilities with a Weighted FOON

Having established FOON as an ideal knowledge representation for robots, especially for deriving novel manipulation sequences through knowledge retrieval, we now explore its use with physical robots. Ideally, a robot can be programmed with the necessary skills to solve problems on its own. A robot using FOON would need to have several components with its knowledge representation, such as perception modules, programmed motion primitives and skills, and both logical and probabilistic formulations of knowledge [3]. To perfectly design a robot that can understand its actions and work in human environments, however, is an exceptionally daunting task. For one, the variability of the environment in which robots work is very dynamic and is likely to feature objects of different shapes and sizes, while also varying in the position of objects. Secondly, robot motions are not guaranteed to be 100% reliable and can fail occasionally. A robot's capability to perform human-like manipulations heavily depends on how it is made; features such as the type of end-effector it has (e.g. what type of gripper it uses, how many fingers it has, etc.), the number of degrees of freedom and joints it has, and the freedom (or lack of) to navigate the environment in search for items it requires. For instance, an Aldebaran NAO robot (which we have used in our experiments) is limited to manipulating lightweight objects; therefore, we address two related questions in this work: 1) how can we reflect the robot's capabilities in FOON? and 2) how can a robot, despite its limited capabilities, use FOON for problem solving?

To address the first question, we introduce weights to FOON that reflect the robot's likelihood of completing an action without failure. Previously, motions were naively assumed to all be 100% reliable in execution, but this does not match the reality of real robots. Weights would be set for robots of different types to reflect their ability to perform certain manipulations. The second question we posed is a very difficult challenge to solve, yet we believe that robots should

A preliminary draft of this chapter was published in [133].

still rely on FOON for problem solving despite their possible limitations. To circumvent this, we have simplified the problem of robotic programming to a human-robot collaboration (HRC) problem, where we can leverage available resources or capabilities of the robot while introducing collaboration with a human assistant. HRC is an ongoing research area that focuses on human and robot interaction to achieve a common goal [134, 135, 136, 137] and has been extensively studied for social interaction [138, 139, 140, 141, 142], coordinated tasks [143, 144, 45, 145] rehabilitation [146, 147], care for the elderly or disabled [148, 149, 150], and many other fields.

Coupled with weights and motivated by previous works on HRC, we consider problem solving as a collaborative effort between a robot and its assistant. In this case, the human acts as an assistant to the robot who has the knowledge to complete the task; given a goal, the robot determines the best course of action through task tree retrieval and collaborates with the human to solve the problem posed to the robotic entity. This not only makes things easier for the human person in reducing the complexity of solving the task (in comparison to doing it on his/her own), but it also improves the chances of the robot succeeding in task tree execution.

7.1 Integrating Weights into a FOON

Up to this point, we have yet to evaluate how a robot can use FOON for task planning. Previously in [2, 103] (and in preceding chapters), all motions were considered to have equal weights in a FOON, implying that all motions can be executed by any robot without failure. Furthermore, any robot should be able to perform the manipulations as well as any other robot or even humans. However, this does not match the reality of current technology since robots come in different shapes and sizes, meaning that they may not all execute manipulations equally in terms of precision. As much as we would like to program any robot to perform any motion, it is difficult to achieve human-like dexterity as observed in demonstrations. For these reasons, we introduce weights in FOON to indicate how challenging a manipulation is to perform. Weights in this paper reflect the robot's *success rate* of performing a given action. Success rate weights (as percentages) are assigned to each functional unit's motion node and are based not only on the manipulation type, but also on the objects contained within the functional unit. In Figure 7.1, success rate weights ranging

between 0 and 1 are assigned to each functional unit. Values are based on: 1) physical capabilities of the robot, 2) past experiences and ability to execute actions, and 3) the tools or objects that the robot needs to manipulate. To guarantee that a robot can perform such motions, weights can be used as heuristics in knowledge retrieval; even though several robots will be equipped with the same universal FOON (meaning they share knowledge from demonstrations), different weights will be assigned to them based on the robot's attributes, which can potentially result in very different task trees. Hence, it is important to note that weights must be defined for each robot. For instance, a small robot like Aldebaran's NAO cannot reliably chop vegetables since it cannot exert the force needed to cut in addition to lacking the dexterity to do so properly.

7.1.1 Deriving Weights for FOON

We can empirically determine representative weights for a robot, where, given a manipulation task, we measure the frequency of successful manipulation trials. It is important to note that when conducting these experiments, one should vary the attributes of the tools or ingredients the robot is manipulating to better capture the conditions in which a robot can sufficiently perform those motions. However, this is not a trivial problem, as motions are likely to have many variables or parameters to tune and learn; for example, when learning to scoop with a spoon, several parameters can be tuned such as where the tool is grasped, the weight of the content in/on the spoon, and the matter or substance that is being scooped. Therefore, to simplify this, we assign estimated weights motions based on our experiences in teaching the robot to perform certain motions in our experiments. Motions that cannot be executed by a robot were assigned a success rate of 0.01 (or 1%), while other motions would be assigned higher values which varies between 0.8 and 0.95 (80 - 95%). Overall, a robot's capabilities to perform tasks in FOON should be based on its perception, strength, dexterity, and reach within its workspace.

7.2 A Weighted Knowledge Retrieval

A FOON can not only be used for representing knowledge, but it can also be used by a robot for problem solving. As discussed in Chapter 5, given a problem defined as a goal, using task tree

retrieval, a robot can obtain a subgraph that contains functional units it needs to follow to solve it. The searching procedure is driven by a list of items in its environment (i.e. the kitchen), which is used to identify ideal functional units based on the availability of inputs to these units. This algorithm is motivated by typical graph-based depth-first search (DFS) and breadth-first search (BFS): starting from the goal node, we search for candidate functional units in a depth-wise manner, while for each candidate, we search among its input nodes in a breadth-wise manner to determine whether or not they are in the kitchen. A subgraph that is obtained from knowledge retrieval is called a *task tree*. A task tree differs from a regular subgraph, as it will not necessarily reflect a complete procedure from a single human demonstration. Rather, it will leverage knowledge from multiple sources to produce a novel task sequence.

However, the naive algorithm does not consider the weights we have added to FOON. In this section, we introduce a different approach to finding the ideal task tree based on success rates, which accounts for every combination of functional units that can be used to solve the problem.

7.2.1 Finding the Optimal Tree

The naive algorithm considers the availability of objects in the robot's environment to determine the best course of action to take in achieving a goal. Knowing what items are in its environment allows us to select steps that can be executed without having to acquire missing items. However, as with all greedy algorithms, this algorithm is not likely to find the optimal course of action. In order to find the task tree with optimality in mind based on weights, we need to explore all possible paths to a given goal node. The objective of this weighted algorithm is to build a tree whose nodes can be explored in a depth-wise manner to find all possible combinations of functional units and picking the path with the highest overall rate of success. All paths to a target object will be given as a tree structure whose nodes contain a combination of functional units needed to make its parent node. We refer to these trees as *path trees*.

In detail, the algorithm (shown as Algorithm 3) works as follows: first, we define a goal node N_{Goal} to the robot. Path tree root nodes (listed in R) comprise of individual functional unit that contains N_{Goal} as output. Initially, these path tree root nodes are appended to a list of path tree

nodes T. Once these root nodes have been identified, over each node t in T, we create and add new tree nodes to T based on their inputs. We iterate for each input object node (in N_{Input}) and identify functional units $FU_{candidate}$ that produce them (i.e. they contain them as output in N_{Output}). For each of these inputs, their $FU_{candidate}$ is added to a list $L_{candidates}$, which is then appended to a list L_{prelim} that covers units for all inputs. When identifying candidates, we will encounter two cases: there may be several functional units that must be executed with other units to create all necessary input objects (non-mutually exclusive events), or there may be multiple candidate functional units that create each input object to choose from (mutually exclusive events). These can be likened to the "AND" and "OR" conditions. Therefore, path tree nodes of depth 1 or higher are not necessary to contain a single functional unit. Using L_{prelim}, we then compute the Cartesian product of candidates to create new path tree nodes for each product set of functional units S in $S_{Cartesian}$; each product set will meet input object requirements of the current node t. These new nodes are then added as children of t and appended to T. The connection between a parent and child node lies in the overlapping of input objects of the parent with the outputs of the child. The propagation of path trees continues until we have identified all of the objects needed to solve the manipulation problem (or simply, until we can no longer add new leaf nodes).

Once all dependencies are met, we then perform depth-first search to find each individual path P from the root nodes (kept in R) to the leaves. Each path will cover all possible functional units that can be followed to solve the given goal. The algorithm from [2] will likely produce one of these paths, but as emphasized before, it is not likely to be the optimal path in terms of success rates. We can use the results from Algorithm 3 to reduce the search space using available items. With the inclusion of weights as success rates for each functional unit in FOON, the optimal task tree would be determined by multiplying the robot's success rate for each action (i.e. functional unit) among all path trees. For example, the total success rate for the sequence in Figure 7.1 is equal to 6.859e-7%. Although this is very low, we can improve the chance of a robot successfully performing a given task through the assistance of another robot or human.

7.2.2 Analysis of the Path Tree Retrieval Algorithm

As opposed to the greedy variant introduced in Chapter 5, this algorithm builds a dependency tree that shares some similarity to AND/OR graphs and uses the tree to identify all possible paths. The dependency tree is found by searching for all variations of functional units that need to be executed (which can be considered as subgoals) to meet a final goal; as a result, this algorithm is considered to be an exponential problem since there may be different variations of paths that can be derived due to the possibly cyclical nature of FOON graphs. However, realistically, the amount of functional units that may be searched for a given end node will not be significant enough to warrant high real-time complexity. The depth-first search that is performed on the final dependency tree is done on all identified roots and performs in $O(|E| \cdot |T|$, where $|E|$ are the number of edges in the tree and $|T|$ is the number of path tree nodes in the dependency tree. We may perhaps consider a dynamic programming approach to make this algorithm run faster in real-time, or better yet, we could also take a similar approach from the task tree retrieval algorithm by using the objects and state of the environment as a cue to refine path tree generation.

7.3 Human-Robot Collaboration

With the addition of weights that reflect the difficulty in executing motions, we can plan while keeping the robot's capabilities in focus. However, because of the overall complexity of human motions as seen in demonstrations, a robot is not guaranteed to perform the same manipulations as well on its own; it would be difficult to program certain manipulations or perhaps the robot is not built to the task. Instead of allowing the robot to act on its own at the risk of failing, it would be best for a robot to collaborate with another entity to improve its chances of successfully solving the problem. This entity can either be another robot or a human assistant who can step in to perform certain actions in its stead. In this section, we will talk about how manipulations can be executed in a collaborative way with the help of a human assistant.

Algorithm 3: Path Tree Retrieval – Searching for all possible task trees

Let N_{Goal} be the goal object node
Let T be list of path tree nodes, R be list of roots of T
{*Find the root functional units for all paths:* }
for all functional units FU_i in G_{FOON} **do**
  if N_{Goal} in N_{Output} of FU_i **then**
    Add FU_i to R and T as path tree node
  end if
end for
{*For all path tree roots, build its dependency tree:* }
for all path tree nodes t in T **do**
  $L_{prelim} = \{\}$
  for all FU_t in t **do**
    for all nodes N_{Input} in FU_t **do**
      $L_{candidates} = \{\}$
      {*Find all candidate units that make each input:* }
      for all functional units FU_i in G_{FOON} **do**
        if N_{Input} in N_{Output} of FU_i **then**
          $FU_{candidate} = FU_i$
        end if
        {*Check if node has not already been visited:* }
        if $FU_{candidate}$ not *ancestor*(t) **then**
          Add $FU_{candidate}$ to $L_{candidates}$
        end if
      end for
      Add $L_{candidate}$ to L_{prelim}
    end for
  end for
  {*Build path tree nodes for all unit combinations:* }
  $S_{Cartesian} = cartesian\_product(L_{prelim})$
  for all ordered sets S in $S_{Cartesian}$ **do**
    Create new path tree node t_{new} containing S
    Set parent of t_{new} as current path tree node t
    Add path tree node t_{new} to T
  end for
  Remove node t from the list T
end for
{*Perform DFS on R to find all task trees:* }
for all path tree nodes t in R **do**
  for all paths P found from $DFS(t)$ **do**
    Print functional units in P
  end for
end for

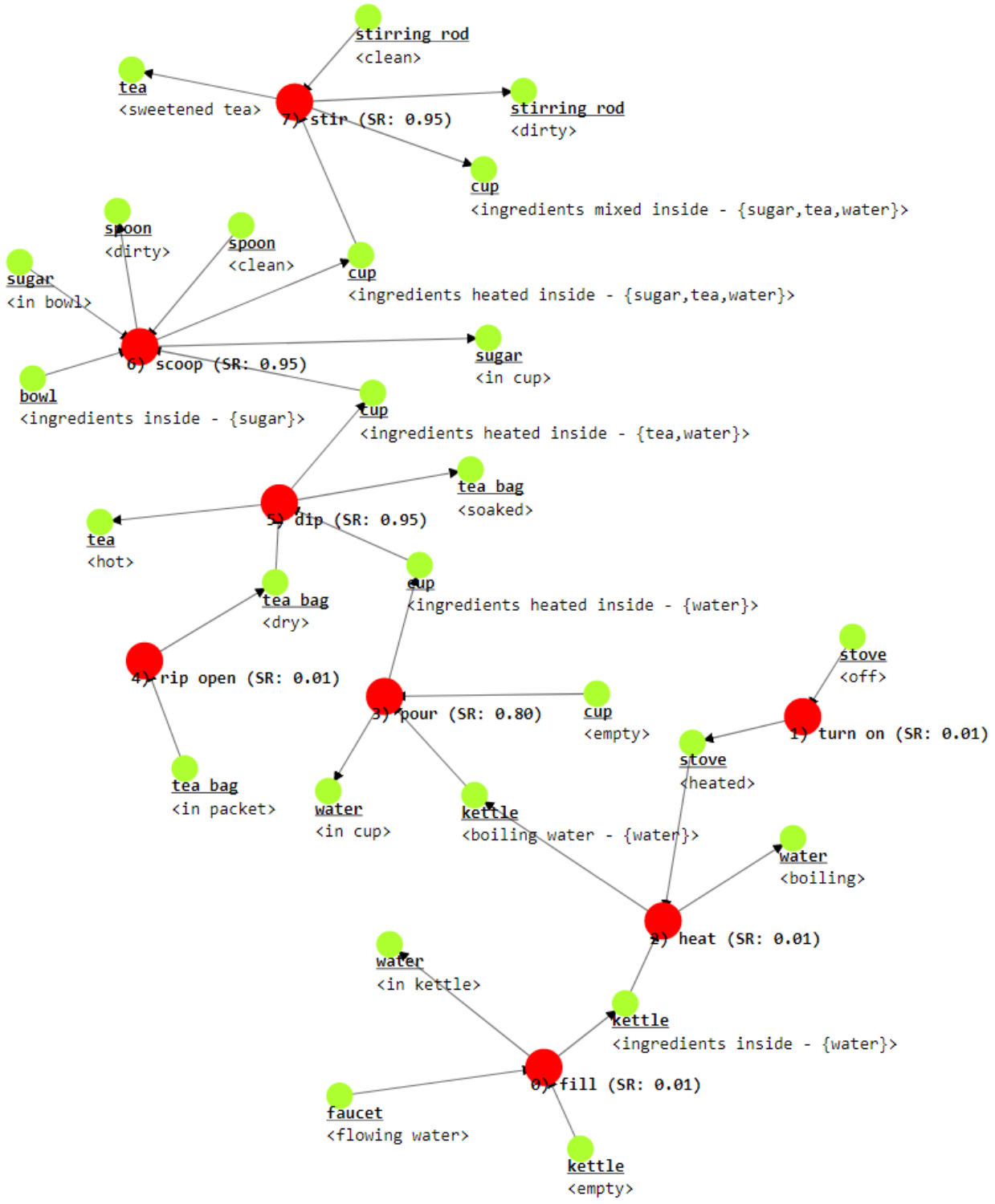


Figure 7.1. Illustration of a weighted subgraph for the activity of making tea. The overall success rate for this subgraph is 6.859e-7% without the involvement of an assistant, which is drastically too low to ever occur without failure.

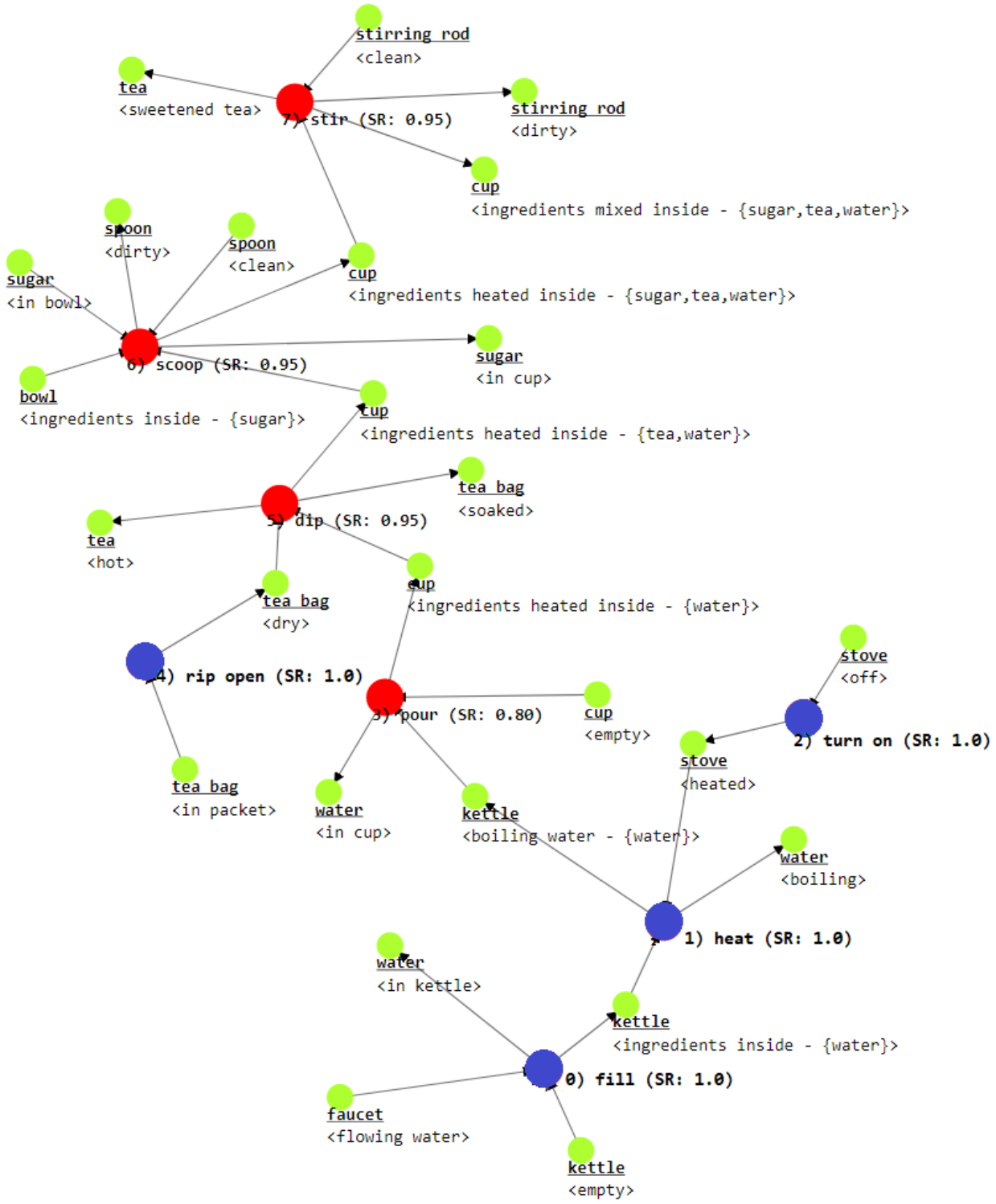


Figure 7.2. Illustration of a weighted subgraph for the activity of making tea, where $M = 4$. The overall success rate for this subgraph with a human assistant increases to 68.59% (since the human will perform the M lowest steps), which is high enough for the robot to perform the task.

7.3.1 Human-assisted Manipulations

With the alternative retrieval algorithm, we can obtain novel task trees for different combinations of methods in a universal FOON. However, certain trees must be eliminated due to the robot's inability to accomplish the required manipulations for all actions described in those task trees; even the execution of the best task tree can still result in failure. Aldebaran's NAO robot, for instance, can only manipulate small, light objects; when compared to larger robots such as the PR2 or Baxter, it cannot perform complex manipulations due to its limited workspace and body configuration. Equally important is its limited mobility to navigate its surroundings since its workspace is very small. To remedy this, we can involve a human assistant in manipulation problems. The human assistant, depending on his/her ability to contribute to the task, can identify the number of steps out of the total number of steps in a task tree that he/she is able and willing to perform to cooperatively solve the problem.

As input to the path tree retrieval, the human can indicate the number of steps as a value M, which cannot exceed the length of the task tree N minus 1 step (as an involvement where N is equal to M means that the human will perform the entire task with no robot assistance in its manipulations). If M is 0, there will be no human involvement in achieving the desired goal. The total success rate of a given path P is denoted by the multiplication of all success rate weights among all functional units in P, which can be likened to the joint probability that all actions are successfully executed. Based on different values of M, for each P, the success rates would be increased by allocating the M lowest units to the human. For each human-assisted step, the success rate would change to 100% by default, for the sake of this paper, unless the human assistant's ability to perform the action is impaired in any way. It is up to the user to determine the degree of involvement he/she is willing to put into an activity, which realistically depends on the person's health/condition, mood, age, skill set, availability and other factors important in HRC tasks. If the human user does not provide a value for M, the optimal value of M can also be determined by the robot; this is done by finding the tree whose success rate at some value of M does not significantly improve over the prior value $M - 1$. In Figure 7.2, the success rate for tea-making increases with assistance; the success rate increases from 6.859e-7% to 68.59%, which

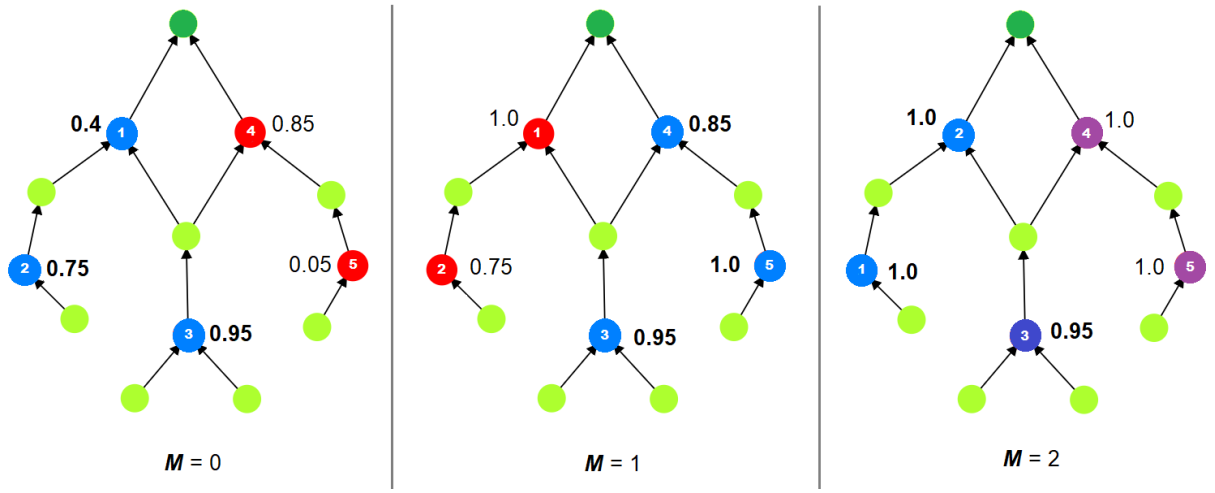


Figure 7.3. An example of how task tree retrieval results can change depending on value of M. As M changes, the total success rate of each path to a goal changes, and thus the ideal task tree obtained differs. The ideal task tree is highlighted in blue, and the end goal is highlighted in dark green. For $M = 0$, the path of functional units {1, 2, 3} will be preferred over the path {3, 4, 5} (28.5% versus 0.8075% chance of success); however, for $M = 1$, the path of units {3, 4, 5} would have a higher weight than the former path (80.75% versus 71.25%). When $M = 2$, we can pick either {1, 2, 3} or {3, 4, 5} as a task tree with a 95% success rate. Here, the two candidate task trees are highlighted in blue and purple, sharing a common unit highlighted in indigo.

is high enough to execute to its entirety. The robot may still fail its manipulations, but it will not have to worry about performing those that are not programmed in its primitives. In the task tree execution phase, the robot will perform its delegated actions, and the remaining M steps are given as instructions to the assistant on how to perform them on the robot's behalf.

We illustrate an example in Figure 7.3 that shows how candidate task trees are weighed against one another and how the total success rate can change between a pair of trees when there is human involvement. As the value of M becomes higher, the ideal task tree changed within trees and caused a significant improvement in the overall success rate of the task (from 28.5% to 95%). However, we can probably make a reasonable trade-off with $M = 1$ rather than $M = 2$ since it should demand less effort from the human assistant.

7.4 Experimental Results

In our experiments, the aim is to show that we can significantly improve robot task manipulation performance through human-robot collaboration within the task planning and execution phases. To demonstrate this, we show that a robot can acquire the ideal task tree for execution, delegate commands to the human assistant, and successfully obtain the goal product for varying levels of involvement. We use Aldebaran's NAO robot to execute manipulations needed to complete the tasks of making tea, mashed potatoes, and ramen noodles. Different variations of preparing each dish (defined as several subgraphs) were merged together into a single, universal FOON, which was then provided to the algorithm to identify different candidate paths for preparing these items and to illustrate how functional units are selected based on success rates. The functional units contained in this universal FOON are all executable by both robot and human and we assume that all items are present in the environment for use. Because the NAO robot itself is very small, its physical capabilities are limited to using smaller versions of items, and furthermore, certain manipulations are very difficult to capture and replicate. Under these circumstances, the robot can greatly benefit from human participation in the task tree execution phase. Certain parts of the tasks, such as heating containers to obtain hot water, cannot be left to the robot to perform; for such motions, their nodes were assigned a very low success rate of 1% to reflect how impossible they are for the robot to do on its own. However, for those motions executable by the robot, we assign higher rates based on our confidence in the robot performing the programmed motion primitives. The task trees obtained through the weighted retrieval approach, along with demonstrations of the robot performing each of these trees, can be viewed in the supplementary material here[1].

7.4.1 Finding the Optimal Task Tree for NAO

First, we show that we can obtain optimal task trees suitable for the NAO robot to prepare tea, mashed potatoes, and ramen noodles. In order to improve the overall success rate of each activity, the task tree algorithm is expected to iterate through several values of M to then determine the optimal M that balances the effort performed by the robot as well as the human assistant. We show

[1]Video demonstrations can be found at the following link: http://www.foonets.com/human-robot.html

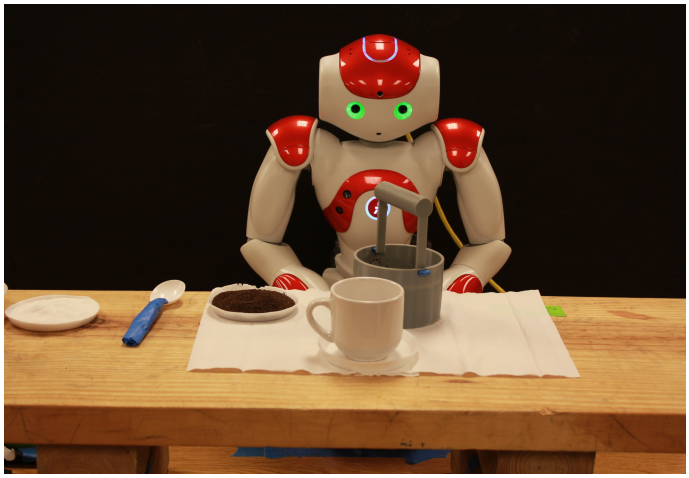


Figure 7.4. Our experimental setup for demonstrating how a weighted FOON and HRC can be used with the NAO robot. NAO is performing the tea-making task using motor primitives, which were taught by demonstration.

the best overall success rates in the graph shown in Figure 7.5 to show how success rates increased as we increased M. As observed from the numbers, the chances of success significantly improve as more steps are delegated to the human assistant. Based on the success rates assigned to the NAO robot's universal FOON, the values of M that were ideal for balanced human-robot manipulations were $M = 1$, $M = 2$, and $M = 3$ for the tasks of mashed potatoes, ramen noodles, and tea-making respectively, as even though some of the robot's primitives have questionably low success rates, it will still be able to execute the task tree on its own. Within the supplementary material, the task trees contain the same number of units labelled as "human-executable" as M.

7.4.2 Executing the Optimal Task Trees

Secondly, we show that we can perform these actions successfully using human-robot collaboration. The NAO robot is programmed to execute certain motions as described in a task tree's motion nodes. Since the objective of this work is to demonstrate the use of a universal FOON in task planning, each motion skill/primitive that can be taught to the robot (such as pouring, scooping, or stirring) are learned by manually recording trajectories to simplify the process of programming the robot and to reduce the complexity of the problem space. We also do not use any sensors nor vision systems for manipulation, as there is no need for object detection. Nevertheless, the

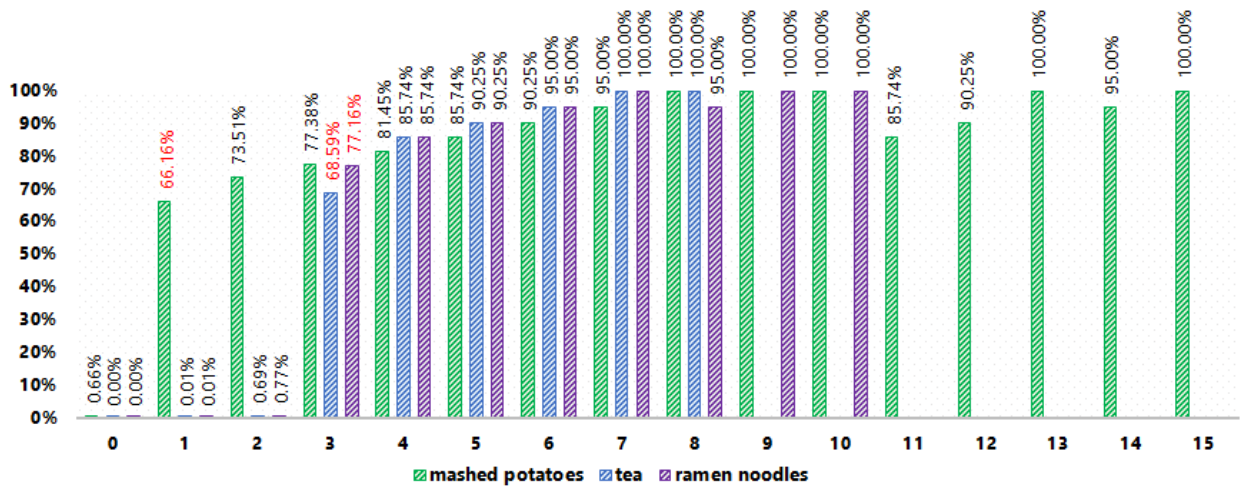


Figure 7.5. Graph showing the gradual improvement in success rates (y-axis) as M (x-axis) increases (best viewed in colour). Sudden drops between M signifies that other paths are considered that exceed the length of M, resulting in a completely human tree (e.g. for values $M = 10$ and $M = 11$, the best potential path trees are different in length). Bars are omitted for values of M that exceed the length of a task tree. The values in red indicate the path tree used in Section 7.4.2.

execution of the entire sequence is determined by the order in which the actions are sequenced in the acquired task tree, meaning that the NAO robot was programmed to perform the activities modularly. In the supplementary material, we provide video demonstrations of the execution of those actions shown in each tree and show how they are carried out with respect to the ideal value of M. Without human involvement, the NAO robot attempts to execute the task tree but ends up failing once it encounters the motion it does not know how to perform (which is reflected by a success rate of 1%); however, with human involvement, the robot can finish all of the tasks and produce the final product. In some cases, we did observe that the motion primitives of the robot can fail, rendering the entire sequence as a failure. As future work, we would like to include sensors or behaviour that allow the robot to determine when it has failed a particular action and to determine what it needs to do to recover from the failed action. Even without its own notion of failure, the robot can supplement this through human interaction by communicating with the assistant to determine whether it should perform the action again.

7.5 Future Work and Ideas Yet Explored

In this section, I will address certain shortcomings to the weighted FOON approach and suggestions that can be considered for future development of FOON for robot task planning.

7.5.1 Determining Weights for Robots

As mentioned before, weights would have to be learned for each robot that will use FOON as its knowledge representation for task planning. However, this itself is a very challenging task due to the high variability of parameters that can be tuned or adjusted when performing motions. For instance, properties such as object weight, texture and size can all vary the success rate of manipulation; in addition, weights cannot be allocated based on motion type alone since a specific combination of input objects may make a manipulation harder than in other instances. Therefore, a more efficient way of measuring weights would need to be standardized. One possible idea can draw from the Million Object Challenge [151], where multiple robots can be set to learn manipulations in parallel continuously and remotely across multiple robots (although this will require many robots to perform training in parallel). We can possibly simplify this by gathering robots of different sizes and architectures and determining what manipulations are best suited for each system; an intelligent kitchen or other household environment would thus rely on several types of robots.

7.5.2 Improving the Path Tree Retrieval Algorithm

The path tree retrieval algorithm is exponential in nature since we are trying to find all variations of sets of functional units that produce a specific goal. If we do not have much variation in performing activities, then we are not likely to experience delays in real-time. However, there may be other ways to significantly reduce the searching time; as discussed before, similar to the greedy algorithm to find a single task tree, a list of ingredients or object instances that are currently in the robot's environment may be useful to avoid searching for dependencies of functional units whose objects are not even available in the first place. However, this may be needed to build from the

top-up since we will not necessarily know if those available objects make any impact until we reach to them in the searching process.

7.5.3 Robot-Robot Collaboration

Following the previous section, multiple robots can be used for learning weights and for executing manipulations that they are best suited for (i.e. they can execute these motions with very low failure rate). Instead of relying on a human assistant to perform motions that a single robotic system cannot perform, we should instead investigate how we can have multiple robots work together to solve a common goal. This problem would require some modification to the task planning, scheduling and execution phases since we would need to plan for manipulations that may be parallel and ensure that the order of sequences are preserved. Robots would need to communicate with one another about when it begins to execute a specific action and when it has completed that operation. Due to the lack of an additional robot that is easier to program, to teach motions to, and to work with existing code, we could not explore how this all can be done.

Chapter 8: Concluding Remarks

To conclude, in this dissertation, we proposed different representations that can be adopted by roboticists and researchers for robotic manipulation and understanding. First, we talked about the functional object-oriented network (FOON), which is a graphical representation of activities as a bipartite network. A robot using FOON will be equipped with the knowledge it needs to solve problems using items that lie in its environment. One key feature of FOON is that a FOON can continuously grow with new concepts through merging of new demonstrations (as subgraphs) or through other proposed techniques that use semantic similarity as a basis for creating new functional units. Finally, we explored how FOON can be used with a real robotic system for problem solving through path tree retrieval to determine the optimal course of action based on success rates. Although we may be limited in resources (i.e. with a higher functionality robot), we showed how a robot with limited physical capabilities or functions can still use FOON to execute task sequences through the aid of a human assistant. Ideally, we would want to have robots that can perform tasks on their own, so the next possible step is to develop robots that can use FOON without human assistance (at least to the extent proposed before). To do so, we could possibly program multiple robots to collaboratively solve a manipulation problem together, should they be unable to execute motions individually or on their own. Another ongoing problem with FOON is creating annotated subgraph files automatically from videos; this itself is a challenge especially for state recognition of objects used in cooking. One avenue not explored heavily is the use of natural language processing to create functional units directly from recipes. Such a procedure would require inference to determine certain states that are not explicitly stated within the text or to determine objects that are not explicitly stated to be used or manipulated in each step.

Second, we talked about the motion taxonomy, another representation but for the purpose of motion representation, analysis and (perhaps) generation. The motion taxonomy was primarily proposed to deviate from the use of natural language labels that suffer from ambiguity, where it is challenging to derive adequate labels in motion classifiers or other representations (such as FOON) for manipulations. We argued that motion codes, which are binarized representations of motions – where each bit or groups of bits represent a particular characteristic of the manipulation –, serve as better embedded representations than others that are derived from natural language (viz. Word2Vec) or simpler embedding schemes such as one-hot encoding. In addition to this, motion codes capture similarities based on real recorded data (from [1]), where force readings for several motion types naturally cluster together in a similar way to motion codes. There is still a lot of work to be done in developing the taxonomy and applying it to classification problems, as we have yet to demonstrate the efficacy of motion codes for machine learning. There may also be other features of motions which we have yet to consider for the taxonomy, which may only be identified after further experiments with motion codes. Once established, we can then investigate how this representation can be used to generate motion trajectories using already learned primitives.

It is important to note that a robot cannot simply rely on these two representations to perform manipulation tasks; as mentioned in [3], a robot will require several components along with its knowledge representation (which can be thought of as a logical formalism of knowledge – FOON addresses this through descriptions of objects and states and transitions between such states through manipulations) to be able to execute tasks on its own. Aside from the grounding of logical statements as a representation, FOON addresses the necessary ability to allow a robot to continuously learn from experiences, where new videos can be annotated and merged to the universal FOON, and a definition of what the robot is expected to do and use to solve problems. However, a robot would still need components such as perception modules and a belief system (to reason based on what the robot believes or understands based on perception and action) to work autonomously. Although FOON can be used in a human-robot collaborative way, we still need to develop a robot (or robots) that do not require the help of humans. We may either build robots can use FOON as a centralized knowledge base to work together in unison or we can program a single,

capable robot to solve problems on its own. In the future, we would need to further investigate these two possibilities. Ideally, we would want to achieve the second scenario, but it would require us to identify a robot that can perform all of the motions in FOON. In addition, we will need to design a kitchen environment that is navigable by robots. Overall, the promise of building service robots for the home are becoming more and more closer to reality, and following the design of intelligent behaviour using knowledge representation and reasoning allow us to develop effective autonomous agents that behave safely in predictable or explainable ways.

References

[1] Yongqiang Huang and Yu Sun. A dataset of daily interactive manipulation. *The International Journal of Robotics Research*, 38(8):879–886, 2019.

[2] David Paulius, Yongqiang Huang, Roger Milton, William D. Buchanan, Jeanine Sam, and Yu Sun. Functional Object-Oriented Network for Manipulation Learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2655–2662, Daejeon, South Korea, 2016. IEEE.

[3] David Paulius and Yu Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 118:13–30, 2019.

[4] Shaogang Ren and Yu Sun. Human-object-object-interaction affordance. In *Workshop on Robot Vision*, 2013.

[5] Yu Sun, Shaogang Ren, and Yun Lin. Object-object interaction affordance learning. *Robotics and Autonomous Systems*, 2013.

[6] Yun Lin and Yu Sun. Robot grasp planning based on demonstrated grasp strategies. *The International Journal of Robotics Research*, 34(1):26–42, 2015.

[7] James J. Gibson. The theory of affordances. In R. Shaw and J. Bransford, editors, *Perceiving, Acting and Knowing*. Hillsdale, NJ: Erlbaum, 1977.

[8] G. Rizzolatti and L. Craighero. The mirror neuron system. *Ann. Rev. Neurosci.*, 27:169–192, 2004.

[9] G. Rizzolatti and Craighero L. Mirror neuron: A neurological approach to empathy. In Jean-Pierre Changeux, Antonio R. Damasio, Wolf Singer, and Yves Christen, editors, *Neurobiology of Human Values*. Springer, Berlin and Heidelberg, 2005.

[10] E. Oztop, M. Kawato, and M. Arbib. Mirror neurons and imitation: a computationally guided review. *Epub Neural Networks*, 19:254–271, 2006.

[11] G. Di Pellegrino, L. Fadiga, L. Fogassi, V. Gallese, and G. Rizzolatti. Understanding motor events: A neurophysiological study. *Exp Brain Res*, 91:176–80, 1992.

[12] V. Gallese, L. Fogassi, L. Fadiga, and G. Rizzolatti. Action representation and the inferior parietal lobule. In W. Prinz and B. Hommel, editors, *Attention and Performance XIX. Common mechanisms in perception and action*. Oxford University Press, Oxford, 2002.

[13] E.Y. Yoon, W.W. Humphreys, and M.J. Riddoch. The paired-object affordance effect. *J. Exp. Psychol. Human*, 36:812–824, 2010.

[14] A. M. Borghi, A. Flumini, N. Natraj, and L. A. Wheaton. One hand, two objects: emergence of affordance in contexfts. *Brain and Cognition*, 80(1):64–73, 2012.

[15] S. Thill, D. Caligiore, A.M. Borghi, T. Ziemke, and G. Baldassarre. Theories and computational models of affordance and mirror systems: An integrative review. *Neuroscience and Biobehavioral Reviews*, 37(3):491–521, 2013.

[16] Hannah Barbara Helbig, Jasmin Steinwender, Markus Graf, and Markus Kiefer. Action observation can prime visual object recognition. *Experimental Brain Research*, 200(3-4):251–258, 2010.

[17] Michael S Ryoo. Human activity prediction: Early recognition of ongoing activities from streaming videos. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1036–1043. IEEE, 2011.

[18] MS Ryoo, Thomas J Fuchs, Lu Xia, Jake K Aggarwal, and Larry Matthies. Robot-centric activity prediction from first-person videos: What will they do to me? In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 295–302. ACM, 2015.

[19] Yu Cao, Daniel Barrett, Andrei Barbu, Siddharth Narayanaswamy, Haonan Yu, Aaron Michaux, Yuewei Lin, Sven Dickinson, Jeffrey Mark Siskind, and Song Wang. Recognize human activities from partially observed videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2658–2665, 2013.

[20] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating the future by watching unlabeled video. *arXiv preprint arXiv:1504.08023*, 2015.

[21] Eren Erdal Aksoy, Alexey Abramov, Florentin Wörgötter, and Babette Dellen. Categorizing object-action relations from semantic scene graphs. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 398–405. IEEE, 2010.

[22] Eren Erdal Aksoy, Alexey Abramov, Johannes Dörr, Kejun Ning, Babette Dellen, and Florentin Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[23] Muralikrishna Sridhar, Anthony G Cohn, and David C Hogg. Learning functional object categories from a relational spatio-temporal representation. In *ECAI 2008: 18th European Conference on Artificial Intelligence (Frontiers in Artificial Intelligence and Applications)*, pages 606–610. IOS Press, 2008.

[24] Yixin Zhu, Yibiao Zhao, and Song Chun Zhu. Understanding tools: Task-oriented object modeling, learning and recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2855–2864, 2015.

[25] Karinne Ramirez-Amaro, Tetsunari Inamura, Emmanuel Dean-León, Michael Beetz, and Gordon Cheng. Bootstrapping humanoid robot skills by extracting semantic representations of human-like activities from virtual reality. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 438–443. IEEE, 2014.

[26] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. Understanding the intention of human activities through semantic perception: observation, understanding and execution on a humanoid robot. *Advanced Robotics*, 29(5):345–362, 2015.

[27] Karinne Ramirez-Amaro, Michael Beetz, and Gordon Cheng. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, 247:95–118, 2017.

[28] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[29] C. Adam Petri and W. Reisig. Petri net. *Scholarpedia*, 3(4):6477, 2008. revision 91646.

[30] Hugo Costelha and Pedro Lima. Modelling, analysis and execution of robotic tasks using petri nets. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1449–1454. IEEE, 2007.

[31] Hugo Costelha and Pedro Lima. Robot task plan representation by petri nets: modelling, identification, analysis and execution. *Autonomous Robots*, 33(4):337–360, 2012.

[32] Christopher Geib, Kira Mourao, Ron Petrick, Nico Pugeault, Mark Steedman, Norbert Krueger, and Florentin Wörgötter. Object action complexes as an interface for planning and robot control. In *IEEE RAS International Conference on Humanoid Robots*, 2006.

[33] Ronald Petrick, Dirk Kraft, Kira Mourao, N Pugeault, N Krüger, and M Steedman. Representation and integration: Combining robot control, high-level planning, and action learning. In *Proceedings of the 6th international cognitive robotics workshop*, pages 32–41. Citeseer, 2008.

[34] Norbert Krüger, Christopher Geib, Justus Piater, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, et al. Object-action complexes: Grounded abstractions of sensory–motor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011.

[35] Mirko Wächter, Sebastian Schulz, Tamim Asfour, Eren Aksoy, Florentin Wörgötter, and Rüdiger Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 189–195. IEEE, 2013.

[36] Daniel Aarno, Johan Sommerfeld, Danica Kragic, Nicolas Pugeault, Sinan Kalkan, Florentin Wörgötter, Dirk Kraft, and Norbert Krüger. Early reactive grasping with second order 3d feature relations. In *Recent Progress in Robotics: Viable Robotic Service to Human*, pages 91–105. Springer, 2007.

[37] G. D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G Barto. Robot learning from demonstration by constructing skill trees. *Intl J Robotics Research*, 31(3):360–375, 2012.

[38] Hao Dang and Peter K Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1311–1317. IEEE, 2012.

[39] Hao Dang and Peter K Allen. Semantic grasping: planning task-specific stable robotic grasps. *Autonomous Robots*, 37(3):301–316, 2014.

[40] Cipriano Galindo, Juan-Antonio Fernández-Madrigal, Javier González, and Alessandro Saffiotti. Robot task planning using semantic maps. *Robotics and autonomous systems*, 56(11):955–966, 2008.

[41] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Andreas Holzbach, and Michael Beetz. Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3601–3608. IEEE, 2009.

[42] Dejan Pangercic, Benjamin Pitzer, Moritz Tenorth, and Michael Beetz. Semantic object maps for robotic housework-representation, acquisition and use. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4644–4651. IEEE, 2012.

[43] Emanuele Bastianelli, Domenico Bloisi, Roberto Capobianco, Guglielmo Gemignani, Luca Iocchi, and Daniele Nardi. Knowledge representation for robots through human-robot interaction. *arXiv preprint arXiv:1307.7351*, 2013.

[44] Thomas Kollar, Vittorio Perera, Daniele Nardi, and Manuela Veloso. Learning environmental knowledge from task-based human-robot dialog. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4304–4309. IEEE, 2013.

[45] Gabriele Randelli, Taigo Maria Bonanni, Luca Iocchi, and Daniele Nardi. Knowledge acquisition through human–robot multimodal interaction. *Intelligent Service Robotics*, 6(1):19–31, 2013.

[46] Yezhou Yang, Cornelia Fermuller, and Yiannis Aloimonos. Detection of manipulation action consequences (mac). In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[47] Yezhou Yang, Anupam Guha, C Fermuller, and Yiannis Aloimonos. A cognitive system for understanding human manipulation actions. *Advances in Cognitive Sysytems*, 3:67–86, 2014.

[48] Yezhou Yang, Anupam Guha, Cornelia Fermuller, and Yiannis Aloimonos. Manipulation action tree bank: A knowledge resource for humanoids. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 987–992. IEEE, 2014.

[49] Yezhou Yang, Cornelia Fermuller, Yi Li, and Yiannis Aloimonos. Grasp type revisited: A modern perspective on a classical feature for vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 400–408, 2015.

[50] Neil Dantam, Pushkar Kolhe, and Mike Stilman. The motion grammar for physical human-robot games. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5463–5469. IEEE, 2011.

[51] Neil Dantam and Mike Stilman. The motion grammar: Analysis of a linguistic method for robot control. *IEEE Transactions on Robotics*, 29(3):704–718, 2013.

[52] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[53] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and Jose Santos-Victor. Modeling affordances using bayesian networks. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4102–4107. IEEE, 2007.

[54] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Learning object affordances: from sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[55] Bogdan Moldovan, Plinio Moreno, and Martijn van Otterlo. On the use of probabilistic relational affordance models for sequential manipulation tasks in robotics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1290–1295. IEEE, 2013.

[56] Bogdan Moldovan and Luc De Raedt. Learning relational affordance models for two-arm robots. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2916–2922. IEEE, 2014.

[57] Bogdan Moldovan, Plinio Moreno, Davide Nitti, José Santos-Victor, and Luc De Raedt. Relational affordances for multiple-object manipulation. *Autonomous Robots*, pages 1–26, 2017.

[58] Francesca Stramandinoli, Vadim Tikhanoff, Ugo Pattacini, and Francesco Nori. Heteroscedastic regression and active learning for modeling affordances in humanoids. *IEEE Transactions on Cognitive and Developmental Systems*, 2017.

[59] Raghvendra Jain and Tetsunari Inamura. Bayesian learning of tool affordances based on generalization of functional feature to estimate effects of unseen tools. *Artificial Life and Robotics*, 18(1-2):95–103, 2013.

[60] Alexander Stoytchev. Behavior-grounded representation of tool affordances. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3060–3065. IEEE, 2005.

[61] Jivko Sinapov and Alexander Stoytchev. Learning and generalization of behavior-grounded tool affordances. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 19–24. IEEE, 2007.

[62] Verica Krunic, Giampiero Salvi, Alexandre Bernardino, Luis Montesano, and José Santos-Victor. Affordance based word-to-meaning association. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4138–4143. IEEE, 2009.

[63] Hedvig Kjellström, Javier Romero, David Martínez, and Danica Kragić. Simultaneous visual recognition of manipulation actions and manipulated objects. *Computer Vision–ECCV 2008*, pages 336–349, 2008.

[64] H. Kjellström, J. Romero, and D. Kragic. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding*, 115:81–90, 2010.

[65] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 8(Mar):693–723, 2007.

[66] Alessandro Pieropan, Carl Henrik Ek, and Hedvig Kjellström. Recognizing object affordances in terms of spatio-temporal object-object relationships. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 52–58. IEEE, 2014.

[67] Alessandro Pieropan, Carl Henrik Ek, and Hedvig Kjellström. Functional object descriptors for human activity modeling. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1282–1289. IEEE, 2013.

[68] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from rgb-d videos. *The International Journal of Robotics Research*, 32(8):951–970, 2013.

[69] Hema S Koppula and Ashutosh Saxena. Anticipating human activities using object affordances for reactive robotic response. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):14–29, 2016.

[70] Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In *European conference on computer vision*, pages 408–424. Springer, 2014.

[71] Emil Vassev and Mike Hinchey. Knowledge representation for cognitive robotic systems. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 156–163. IEEE, 2012.

[72] Emil Vassev and Mike Hinchey. Knowlang: Knowledge representation for self-adaptive systems. *IEEE Computer*, 48(2):81–84, 2015.

[73] Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai Häussermann, Rob Janssen, JMM Montiel, Alexander Perzylo, et al. RoboEarth. *IEEE Robotics & Automation Magazine*, 18(2):69–82, 2011.

[74] Luis Riazuelo, Moritz Tenorth, Daniel Di Marco, Marta Salas, Dorian Gálvez-López, Lorenz Mösenlechner, Lars Kunze, Michael Beetz, Juan D Tardós, Luis Montano, et al. RoboEarth semantic mapping: A cloud enabled knowledge-based approach. *IEEE Transactions on Automation Science and Engineering*, 12(2):432–443, 2015.

[75] Luis Riazuelo, Javier Civera, and JMM Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[76] Dominique Hunziker, Mohanarajah Gajamohan, Markus Waibel, and Raffaello D'Andrea. Rapyuta: The roboearth cloud engine. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 438–444. IEEE, 2013.

[77] Gajamohan Mohanarajah, Dominique Hunziker, Raffaello D'Andrea, and Markus Waibel. Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, 2015.

[78] Michael Beetz, Moritz Tenorth, and Jan Winkler. Open-EASE. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1983–1990. IEEE, 2015.

[79] Moritz Tenorth and Michael Beetz. KnowRob: Knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4261–4266. IEEE, 2009.

[80] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the knowrob framework. *Artificial Intelligence*, 247:151–169, 2017.

[81] Michael Beetz, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. KnowRob 2.0âĂŤa 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2018.

[82] Dominik Jain, Lorenz Mosenlechner, and Michael Beetz. Equipping robot control programs with first-order probabilistic reasoning capabilities. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3626–3631. IEEE, 2009.

[83] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. CRAMâĂŤa cognitive robot abstract machine for everyday manipulation in human environments. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1012–1017. IEEE, 2010.

[84] Michael Beetz, Dominik Jain, Lorenz Mosenlechner, Moritz Tenorth, Lars Kunze, Nico Blodow, and Dejan Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012.

[85] Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K Misra, and Hema S Koppula. Robobrain: Large-scale knowledge engine for robots. *arXiv preprint arXiv:1412.0691*, 2014.

[86] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[87] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[88] John R Anderson. *Rules of the mind*. Psychology Press, 2014.

[89] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[90] John E Laird. *The Soar cognitive architecture*. MIT press, 2012.

[91] Ron Sun, Edward Merrill, and Todd Peterson. From implicit skills to explicit knowledge: a bottom-up model of skill learning. *Cognitive science*, 25(2):203–244, 2001.

[92] Davis E Kieras and Davis E Meyer. An overview of the epic architecture for cognition and performance with application to human-computer interaction. *Human–Computer Interaction*, 12(4):391–438, 1997.

[93] John E Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive robotics using the soar cognitive architecture. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence, Cognitive Robotics*, 2012.

[94] Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, pages 1469–1474. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[95] KangGeon Kim, Ji-Yong Lee, Dongkyu Choi, Jung-Min Park, and Bum-Jae You. Autonomous task execution of a humanoid robot using a cognitive model. In *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, pages 405–410. IEEE, 2010.

[96] KangGeon Kim, Dongkyu Choi, Ji-Yong Lee, Jung-Min Park, and Bum-Jae You. Controlling a humanoid robot in home environment with a cognitive architecture. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 1754–1759. IEEE, 2011.

[97] David Vernon, Giorgio Metta, and Giulio Sandini. The icub cognitive architecture: Interactive development in a humanoid robot. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 122–127. Ieee, 2007.

[98] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes Von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, et al. The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23(8):1125–1134, 2010.

[99] Serena Ivaldi, Natalia Lyubova, Damien Gérardeaux-Viret, Alain Droniou, Salvatore M Anzalone, Mohamed Chetouani, David Filliat, and Olivier Sigaud. Perception and human interaction for developmental learning of objects and affordances. In *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*, pages 248–254. IEEE, 2012.

[100] J Gregory Trafton, Laura M Hiatt, Anthony M Harrison, Franklin P Tamborello II, Sangeet S Khemlani, and Alan C Schultz. Act-r/e: An embodied cognitive architecture for human-robot interaction. *Journal of Human-Robot Interaction*, 2(1):30–55, 2013.

[101] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[102] A. B. Jelodar, D. Paulius, and Y. Sun. Long Activity Video Understanding Using Functional Object-Oriented Network. *IEEE Transactions on Multimedia*, 21(7):1813–1824, July 2019.

[103] David Paulius, Ahmad B Jelodar, and Yu Sun. Functional Object-Oriented Network: Construction & Expansion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5935–5941, Brisbane, Australia, 2018. IEEE.

[104] A. B. Jelodar, M. S. Salekin, and Y. Sun. Identifying object states in cooking-related images. *arXiv preprint arXiv:1805.06956*, May 2018.

[105] FOON Project Website. http://www.foonets.com. Accessed: 2020-03-16.

[106] Sanjeeth Bhat. A Study of Data Storage Optimization for the Functional Object-Oriented Network. *USF Honors College*, 2019. Submitted as Honors Thesis.

[107] David Paulius. FOON API - BitBucket Repository. https://bitbucket.org/davidpaulius/foon_api/src. Accessed: 2020-02-14.

[108] Bernard Ghanem Fabian Caba Heilbron, Victor Escorcia and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[109] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling Egocentric Vision: The EPIC-KITCHENS Dataset. In *European Conference on Computer Vision (ECCV)*, 2018.

[110] Javier Marin, Aritro Biswas, Ferda Ofli, Nicholas Hynes, Amaia Salvador, Yusuf Aytar, Ingmar Weber, and Antonio Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.

[111] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, USA, 2010.

[112] David Paulius and Yu Sun. Functional object-oriented network: A knowledge representation for service robotics. *The International Journal of Robotics Research (IJRR)*, TBD:TBD, 2020.

[113] Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *CVPR*, pages 1194–1201. IEEE Computer Society, 2012.

[114] David Paulius, Yongqiang Huang, Jason Meloncon, and Yu Sun. Manipulation Motion Taxonomy and Coding for Robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5596–5601, Macau, China, 2019. IEEE.

[115] Mark R Cutkosky. On grasp choice, grasp models, and the design of hands for manufacturing tasks. *IEEE Transactions on robotics and automation*, 5(3):269–279, 1989.

[116] Florentin Wörgötter, Eren Erdal Aksoy, Norbert Krüger, Justus Piater, Ales Ude, and Minija Tamosiunaite. A simple ontology of manipulation actions based on hand-object relations. *IEEE Transactions on Autonomous Mental Development*, 5(2):117–134, 2013.

[117] Ian M Bullock, Raymond R Ma, and Aaron M Dollar. A hand-centric classification of human and robot dexterous manipulation. *IEEE transactions on Haptics*, 6(2):129–144, 2013.

[118] Thomas Feix, Javier Romero, Heinz-Bodo Schmiedmayer, Aaron M Dollar, and Danica Kragic. The grasp taxonomy of human grasp types. *IEEE Transactions on Human-Machine Systems*, 46(1):66–77, 2016.

[119] Yuzuko C Nakamura, Daniel M Troniak, Alberto Rodriguez, Matthew T Mason, and Nancy S Pollard. The complexities of grasping in the wild. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 233–240. IEEE, 2017.

[120] Wei Dai, Yu Sun, and Xiaoning Qian. Functional analysis of grasping motion. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3507–3513. IEEE, 2013.

[121] Bahareh Abbasi, Ehsan Noohi, Sina Parastegari, and Miloš Žefran. Grasp taxonomy based on force distribution. In *Robot and Human Interactive Communication (RO-MAN), 2016 25th IEEE International Symposium on*, pages 1098–1103. IEEE, 2016.

[122] Hamal Marino, Marco Gabiccini, Ales Leonardis, and Antonio Bicchi. Data-driven human grasp movement analysis. In *ISR 2016: 47st International Symposium on Robotics; Proceedings of*, pages 1–8. VDE, 2016.

[123] Yongqiang Huang, Matteo Bianchi, Minas Liarokapis, and Yu Sun. Recent data sets on object manipulation: A survey. *Big Data*, 4(4):197–216, 2016.

[124] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[125] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–317. IEEE, 2007.

[126] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[127] Robyn Speer, Joshua Chin, and Catherine Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *AAAI Conference on Artificial Intelligence*, pages 4444–4451, 2017.

[128] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. Wikipedia2Vec: An Optimized Tool for Learning Embeddings of Words and Entities from Wikipedia. *arXiv preprint 1812.06280*, 2018.

[129] Yongqiang Huang and Yu Sun. Generating manipulation trajectory using motion harmonics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4949–4954. IEEE, 2015.

[130] Yongqiang Huang and Yu Sun. Learning to pour. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7005–7010. IEEE, 2017.

[131] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics, 1994.

[132] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. " O'Reilly Media, Inc.", 2009.

[133] David Paulius, Kelvin Sheng Pei Dong, and Yu Sun. Functional object-oriented network: Considering robot's capability in human-robot collaboration. *arXiv preprint arXiv:1905.00502*, 2019.

[134] Terrence Fong, Illah Nourbakhsh, and Kerstin Dautenhahn. A survey of socially interactive robots. *Robotics and autonomous systems*, 42(3-4):143–166, 2003.

[135] Holly A Yanco and Jill Drury. Classifying human-robot interaction: an updated taxonomy. In *systems, man and cybernetics, 2004 IEEE International Conference on*, volume 3, pages 2841–2846. IEEE, 2004.

[136] Michael A Goodrich, Alan C Schultz, et al. Human–robot interaction: a survey. *Foundations and Trends® in Human–Computer Interaction*, 1(3):203–275, 2008.

[137] Balasubramaniyan Chandrasekaran and James M Conrad. Human-robot collaboration: A survey. In *SoutheastCon 2015*, pages 1–8. IEEE, 2015.

[138] Oussama Khatib. Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26(2-3):175–183, 1999.

[139] Michael Zinn, Oussama Khatib, Bernard Roth, and J Kenneth Salisbury. Playing it safe [human-friendly robots]. *IEEE Robotics & Automation Magazine*, 11(2):12–21, 2004.

[140] A. Edsinger and C. C. Kemp. Human-robot interaction for cooperative manipulation: Handing objects to one another. In *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*, pages 1167–1172. IEEE, 2007.

[141] Kerstin Dautenhahn. Socially intelligent robots: dimensions of human–robot interaction. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 362(1480):679–704, 2007.

[142] Paul E Rybski, Jeremy Stolarz, Kevin Yoon, and Manuela Veloso. Using dialog and human observations to dictate tasks to a learning robot assistant. *Intelligent Service Robotics*, 1(2):159–167, 2008.

[143] Bilge Mutlu and Jodi Forlizzi. Robots in organizations: the role of workflow, social, and environmental factors in human-robot interaction. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, pages 287–294. ACM, 2008.

[144] Christine Bringes, Yun Lin, Yu Sun, and Redwan Alqasemi. *Determining the benefit of human input in human-in-the-loop robotic systems*. IEEE, 2013.

[145] Matthew C Gombolay, Cindy Huang, and Julie A Shah. Coordination of human-robot teaming with human task preferences. In *AAAI Fall Symposium Series on AI-HRI*, volume 11, page 2015, 2015.

[146] Ben Robins, Paul Dickerson, Penny Stribling, and Kerstin Dautenhahn. Robot-mediated joint attention in children with autism: A case study in robot-human interaction. *Interaction studies*, 5(2):161–198, 2004.

[147] Aude Billard, Ben Robins, Jacqueline Nadel, and Kerstin Dautenhahn. Building robota, a mini-humanoid robot for the rehabilitation of children with autism. *Assistive Technology*, 19(1):37–49, 2007.

[148] Henry Kautz, Larry Arnstein, Gaetano Borriello, Oren Etzioni, and Dieter Fox. An overview of the assisted cognition project. In *AAAI-2002 Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care*, page 6065, 2002.

[149] RS Rao, K Conn, Sang-Hack Jung, Jayantha Katupitiya, Terry Kientz, Vijay Kumar, J Ostrowski, Sarangi Patel, and Camillo J Taylor. Human robot interaction: application to smart wheelchairs. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3583–3588. IEEE, 2002.

[150] Kerstin Dautenhahn, Michael Walters, Sarah Woods, Kheng Lee Koay, Chrystopher L Nehaniv, A Sisbot, Rachid Alami, and Thierry Siméon. How may i serve you?: a robot companion approaching a seated person in a helping context. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 172–179. ACM, 2006.

[151] John Oberlin, Maria Meier, Tim Kraska, and Stefanie Tellex. Acquiring object experiences at scale. *AAAI-RSS Special Workshop on the 50th Anniversary of Shakey: The Role of AI to Harmonize Robots and Humans*, 2015.

Appendix A: HOW-TO: Using the FOON API

In the FOON API repository [107], which can be found at the following link[1], we have three (3) main files written in Python:

- *FOON_classes.py* – definition of object-oriented classes to define node types and structures
- *FOON_graph_analyzer.py* – definition of functions that operate on FOON structures
- *FOON_parser.py* – script used to parse subgraph files, make objects, motions and states consistent with regards to labelling, and output index files

Complementary to these Python scripts is a graph visualization tool, which was written in HTML and JavaScript, that can be used for easy viewing and verification of graph files called *FOON-view*. This can also be accessed through our website [105]. Instructions on how to use this tool is provided in Appendix B.

A.1 FOON_classes.py

Having taken an object-oriented approach to programming, the *FOON_classes.py* file defines structures for the API as classes. The following are the main classes typically needed for FOON:

- `Thing` class – a superclass for nodes in the network. `Thing` objects are described by an identifier, a label, and, most importantly, a list of neighbouring nodes. The list of neighbours is important to preserve the adjacency list representation in FOON. In summary, any node in the network is a `Thing`; a `Thing` object can either be an `Object` or `Motion` object. A `Thing` object by default considers an object as level 1 since it only has a type and label.

[1]FOON API Repository - https://bitbucket.org/davidpaulius/foon_api/src

- `Object` class – a class that inherits properties of a generic node (viz. `Thing`). `Object` class instances describe the tools or ingredients that are used in manipulations. The neighbours of an `Object` object instance are always `Motion` object instances when building FOON files by default to preserve the bipartite structure of FOON; however, the only exception to this is the derivation of a one-mode projection of FOON which contains purely `Object` nodes. Like established before in Chapter 2, `Objects` instances are mainly identified by:
  - an *object type* – an integer-type identifier for an `Object` (e.g. label *'avocado'* is ID *O8*).
  - an *object label* – a string-type label that names the type of object that is described by an `Object` instance (e.g. items like *'asparagus'* or *'cutting board'*).
  - a *list of states* – an object's states can be broken down into *identifier*, *label*, and *ingredient composition* based on the state the object is in. As of FOON-100, objects can have multiple states to describe the object in the environment.
  - a *location identifier* and *label* – a recent addition to the `Object` class. These variables describe the location of an object within the activity described by functional units. These variables are optional and can be used to remove states that pertain to the location of an object rather than a physical state of the object. In *FOON-100*, objects do not have these labels assigned to them, and so this is unusable for the time-being.
- `Motion` class – a class that also inherits properties from a generic node (viz. `Thing`). This class describes motion nodes. The neighbours of a `Motion` object instance are always `Object` instances. A universal FOON comprises of many motion nodes that can be of duplicate types as long as its input and output object nodes are unique among all other functional units. Like established before in Chapter 2, motion nodes are mainly identified by:
  - a *motion type* – an integer-type identifier for the motion (e.g. label *'pour'* is ID *M40*).
  - a *motion label* – a string-type label that names the type of manipulation that is described by a `Motion` instance (e.g. manipulations such as 'sprinkle' or 'flip').

- `FunctionalUnit` class – a class describing functional units in FOON. A `FunctionalUnit` object contains several internal lists that mainly keep track of input object nodes, output object nodes, and motion nodes. A `FunctionalUnit` object will contain accessor and mutator functions for input and output objects to iterate through these lists as well as functions to access or modify the motion node assigned to it. In addition to these lists, a `FunctionalUnit` object instance also contains the time-stamps for these manipulations as seen in the source demonstration, the objects' motion identifiers (which describe whether objects are moving or immobile in the manipulation), and a floating success rate, if present in the subgraph.

- `Category` class – a class that describes a wrapper class for representing object categories for *FOON-GEN* creation (i.e. FOON generalization via compression).

- `TreeNode` class – a specialized tree node definition for the path tree retrieval procedure when finding optimal paths in task planning. A `TreeNode` has basic accessor and mutator functions to create a path tree which can iterated through in a depth-first manner to derive each possible path to a goal node.

Both `Object` and `FunctionalUnit` classes have different equality functions based on each hierarchy level (once again, refer to Chapter 2 for what each level means), viz. `equals_lvl3`, `equals_lvl2`, and `equals_lvl1`. The higher the level, the more of an `Object`'s attributes are used when determining whether two object nodes are the same or not. The `FunctionalUnit`'s equality functions rely on the `Object` class's equality functions to decide on whether two functional units are the same or not; they do not consider whether an object is moving or not (based on the motion identifier) since we care more about the coupling of objects that are used or interact for specific actions or purposes. Definitions based on all three hierarchy levels are also given for the output methods for `Object` and `FunctionalUnit` instances as well.

Please refer to the code for more details on these definitions, functions and their implementations. Interested readers may contact me directly for more information about the code or FOON.

A.2 FOON_graph_analyzer.py

The *FOON_graph_analyzer.py* file is the main driving script available in the FOON API, as it contains several key functions for many operations such as to read FOON files, merge subgraphs into a universal FOON, perform task tree retrieval, perform path tree retrieval, and calculate node centrality on objects. This file must be used with the accompanying *FOON_classes.py* file in order for these functions to do anything meaningful. When running this script, users have the option of providing specific arguments that are read on initialization. They are all optional, but they make certain operations quicker (such as the task tree retrieval function). These include the following:

- `--file`, which indicates to the script the name of the FOON file (or path to the file) that will be given as input to the program. This is used in the format `-file=X`, where `X` is the name or path. If this argument is not given, then the program will default to opening another file that is hard-coded in the script; users can modify this to suit their needs.

- `--verbose`, which is a flag to enable verbose mode for the script, thereby printing extra output to the console. This can be very useful to understand what is happening behind the scenes and when/where errors may be encountered.

- `--object`, which is an argument that indicates the *object type* that is to be used as a parameter to task or path tree retrieval function. It is parsed as an integer and used in the format `--object=Y`, where Y is an integer.

- `--state`, which is an argument that indicates the object state type(s) that is to be used as a parameter to task or path tree retrieval function. Since an object may have multiple states, these numbers should be given in the form of a list. For instance, we can provide `--state=[1,44]`, where we provide the criteria for a goal object having two states with types *1* and *44* (whose labels can be identified through the state index file).

- `--help`, which prints out a summary of the various input arguments that have been defined for this script.

A.2.1 Primary Functions in FOON_graph_analyzer.py

In this section, I will highlight a few important functions that I have defined in this script. These functions are based on the concepts discussed in the main chapters of this dissertation.

A.2.1.1 `_constructFOON` function

Perhaps the most crucial function in this entire code, this function populates all list structures in FOON for object and motion nodes and creates functional units. In simple terms, this function iterates through each line of a FOON file (if provided as a parameter; else, it uses the file defined in the program's header) and derives all object and motion nodes described in it. Each functional unit is separated by \\, which the program uses to determine the start and end of a functional unit description. This function works alongside sub-operations `_checkIfNodeExists()` and `_checkIfNodeExists()` to determine whether a node already exists in FOON (to make new connections to existing objects) and to check if a duplicate of a functional unit is present in FOON respectively. When merging multiple files, this function inherently reads multiple files and checks for duplicates, so the merging function (`_createUniversalFOON()`) simply calls upon this function for each unmerged file.

A.2.1.2 `_buildFunctionalUnitMap` function

This function is a fairly new procedure that populates various dictionary structures through the use of sub-procedures: 1) `_buildObjectUnitMap`, which builds dictionaries, for all hierarchy levels, that map every object node to a list of functional units that contain its `Object` instance within its output nodes, and 2) `_buildUnitToUnitMap`, which builds dictionaries that map functional unit X to Y, where the output nodes of X overlap with input nodes of Y. As mentioned in Chapter 5, by "caching" with dictionaries, we can speed up the searching process where it requires looping through the entire FOON to find solutions for goals and subgoals. The same can be said of the path tree algorithm, where we can find overlapping units faster to grow the path tree quicker. Users defining their own functions in the API can perhaps adapt these structures in other analytical operations to simplify their code.

A.2.1.3 `_createUniversalFOON` function

As mentioned before, this function accepts a path to a folder of FOON files that are being merged with the current FOON that is read into the script and runs the `_constructFOON()` function on each file. Each file in the directory are passed into the `_constructFOON()` function as parameters. After running through each file, the new universal FOON file is outputted as a new text file through the `_outputUniversalFOON()` function.

A.2.1.4 `_taskTreeRetrieval_greedy` function

The task tree retrieval algorithm from Chapter 5 is implemented as this algorithm. The first step in this function is to identify all possible goal nodes that match the object and state types passed as parameters to the function (or as arguments to the entire script); this is done to find variants of the goal whose requirements are met by the kitchen items. The next step in this function is then to read a file containing the list of items in the kitchen or environment; if this file is not given as input, then the function is designed to call upon another function `_printStartNodes()`, which identifies all starting nodes in the present FOON. Once these kitchen items are added to the list `kitchen` and the goal node is added to a queue `itemsToSearch`, the search begins. Since the algorithm is not guaranteed to terminate until the goal node becomes present in `kitchen`, the algorithm should only iterate for a limited number of encounters of the goal as the head of `itemsToSearch`. This is controlled by the `depth` variable. The algorithm will either time out, when we exceed `depth`, or it will return a text file with the functional units to the solution task tree. A modified version of this algorithm, `_taskTreeRetrieval_optimal`, takes weights into consideration.

A.2.1.5 `_calculateCentrality` function

This function is used to compute Katz and degree centralities with the input FOON (based on Chapter 3. First, it creates a one-mode projection of FOON objects and then an adjacency matrix of this network. Users must indicate the hierarchy level at which centrality is being computed. The adjacency matrix is then used to calculate Katz centrality values, and objects are simply checked to count the number of neighbours they have for degree centrality. These centrality values are

outputted in files. In order to use this function, NumPy must be installed to use its solver to derive the centrality values for each node.

A.2.1.6 `_expandNetwork` *function*

This function is the main driver for the expansion procedure described in Chapter 6. As inputs to this function, the user should specify the threshold for similarity and the source used to measure relatedness (viz. WordNet [86] or Concept-Net [127]). First, we build an object similarity index file through the `_buildObjectSimilarity`, where we use either WordNet or Concept-Net to compute relatedness as a value, from which we then expand and create new functional units based on the threshold. The expansion also considers ingredients and tries to create new objects while substituting similar items accordingly. When creating the newly expanded universal FOON, the user has the option to use Concept-Net to verify whether objects added through expansion match their assigned states (as per the suggestion at the end of Chapter 6). Once this process is over, the newly expanded network is outputted to a text file. The expanded network can be used as an argument to the code for task tree retrieval. In order to use this function with WordNet, it is required that the user has NLTK [132] installed to download the corpus and to compute relatedness with its built-in Wu-Palmer [131] implementation for WordNet use or *gensim* to read the word embeddings from Concept-Net.

A.2.1.7 `_constructFOON_GEN` *function*

This function is a specialized version of the `_constructFOON` function that creates a generalized FOON based on object categories, which was covered in Chapter 6. It requires a file called 'object_categories.txt' (which is to be provided in the same repository) that defines categories and their member objects since we cannot simply use knowledge bases like we did in the expansion function. Other than the idea of using categories, the function works in the same way as the original version: it reads a text file and creates node objects and structures to represent FOON.

A.3 FOON_parser.py

One important step in the creation of a universal FOON for task planning is annotation. Annotation ideally should be done automatically through the use of activity and video understanding, but currently everything is done manually by researchers working on FOON (including myself). When annotating by hand, it is very likely that there will be errors in formatting, specifically when labelling nodes or in following the strict format of graphs (such as with tab-spacing); similarly, new labels may be added to FOON, but they do not exist in current label indices. Each index file, which we have for object, state and motion labels, has to be constantly updated and reviewed when annotating new subgraphs. To do these two things, following a parsing script created by our former undergraduate researcher William Buchanan, I created an updated version of this script to make annotation easier. The *FOON_parser.py* script simply prompts the user to input the location of (or path to) the inconsistent FOON files to read and iterate through the files, line by line, to build new index files, after which it uses to parse through the FOON files once more to correct the labels and create new, fixed files. Each index file has labels sorted in alphabetical order to make it easy for users to revise them and to see if there are any repeated entries due to some erroneous character inputs. Revised FOON files have a timestamp (reflecting the date they were made) appended to its name to distinguish them from the older files.

Appendix B: HOW-TO: Using the FOON-view Tool

On our project's website [105], we provide a visualization tool called *FOON-view*[1]. This interface was built using JavaScript and HTML by building upon the classes we have already established in Java and Python. The steps to be taken to use this tool for viewing graphs is as follows:

1. First, one must obtain a text file that is annotated in the FOON convention. You may either use subgraphs provided in the API folder or you may annotate a subgraph yourself while using the *FOON_parser.py* script that was discussed in the prior chapter.

2. Second, open your web browser to the provided link. You will see a blank canvas and a button that says *"Choose File"*; click this button and select the text file you wish to visualize.

3. Third, you will be prompted to provide the hierarchy level you wish to view (given as a value from 1 to 3). This prompt is done through a dialog that is presented through the web browser. Please review Chapter 2 for more information on these hierarchy levels.

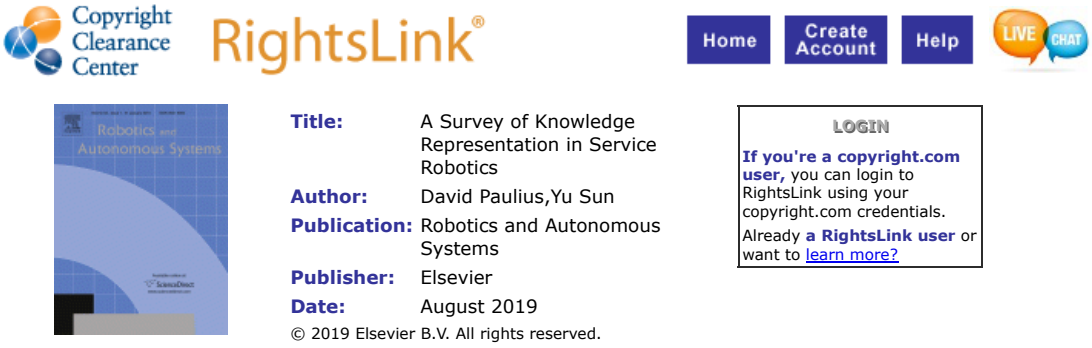
4. Finally, once you have provided the level, you will see the graph in the canvas.

    - Object nodes are denoted by lime-green nodes, which have an object type and object state(s) attributed to them.
    - Motion nodes are denoted by red nodes, which have a motion type attributed to them. These nodes are numbered based on their corresponding functional unit's appearance in the text file.
    - To make the graph smaller/larger, you may zoom in/out using the scroll wheel.
    - You may also move nodes about by clicking and dragging them about the canvas.

[1] FOON Graph Visualization Tool – http://foonets.com/FOON_view/visualizer.html

Appendix C: Copyright Permissions

Permission from [3] for use of content in Chapter 1 is shown below.



Copyright Clearance Center RightsLink®

Home | Create Account | Help | LIVE CHAT

Title: A Survey of Knowledge Representation in Service Robotics

Author: David Paulius,Yu Sun

Publication: Robotics and Autonomous Systems

Publisher: Elsevier

Date: August 2019

© 2019 Elsevier B.V. All rights reserved.

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.

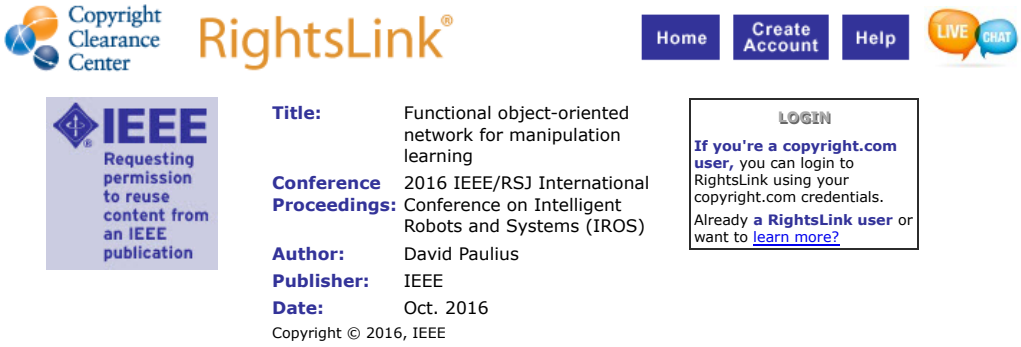
Already **a RightsLink user** or want to learn more?

Please note that, as the author of this Elsevier article, you retain the right to include it in a thesis or dissertation, provided it is not published commercially. Permission is not required, but please ensure that you reference the journal as the original source. For more information on this and on your other retained rights, please visit: https://www.elsevier.com/about/our-business/policies/copyright#Author-rights

BACK | CLOSE WINDOW

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com

Permission from [2] for use of content in Chapters 1, 2, 3 and 5 is shown below.



Copyright Clearance Center RightsLink®

Home | Create Account | Help | LIVE CHAT

IEEE Requesting permission to reuse content from an IEEE publication

Title: Functional object-oriented network for manipulation learning

Conference Proceedings: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Author: David Paulius

Publisher: IEEE

Date: Oct. 2016

Copyright © 2016, IEEE

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.

Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE WINDOW

Permission from [103] for use of content in Chapters 2 and 6 is shown below.

Copyright Clearance Center RightsLink®

Home | Create Account | Help

IEEE Requesting permission to reuse content from an IEEE publication

Title: Functional Object-Oriented Network: Construction & Expansion

Conference Proceedings: 2018 IEEE International Conference on Robotics and Automation (ICRA)

Author: David Paulius

Publisher: IEEE

Date: May 2018

Copyright © 2018, IEEE

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.

Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

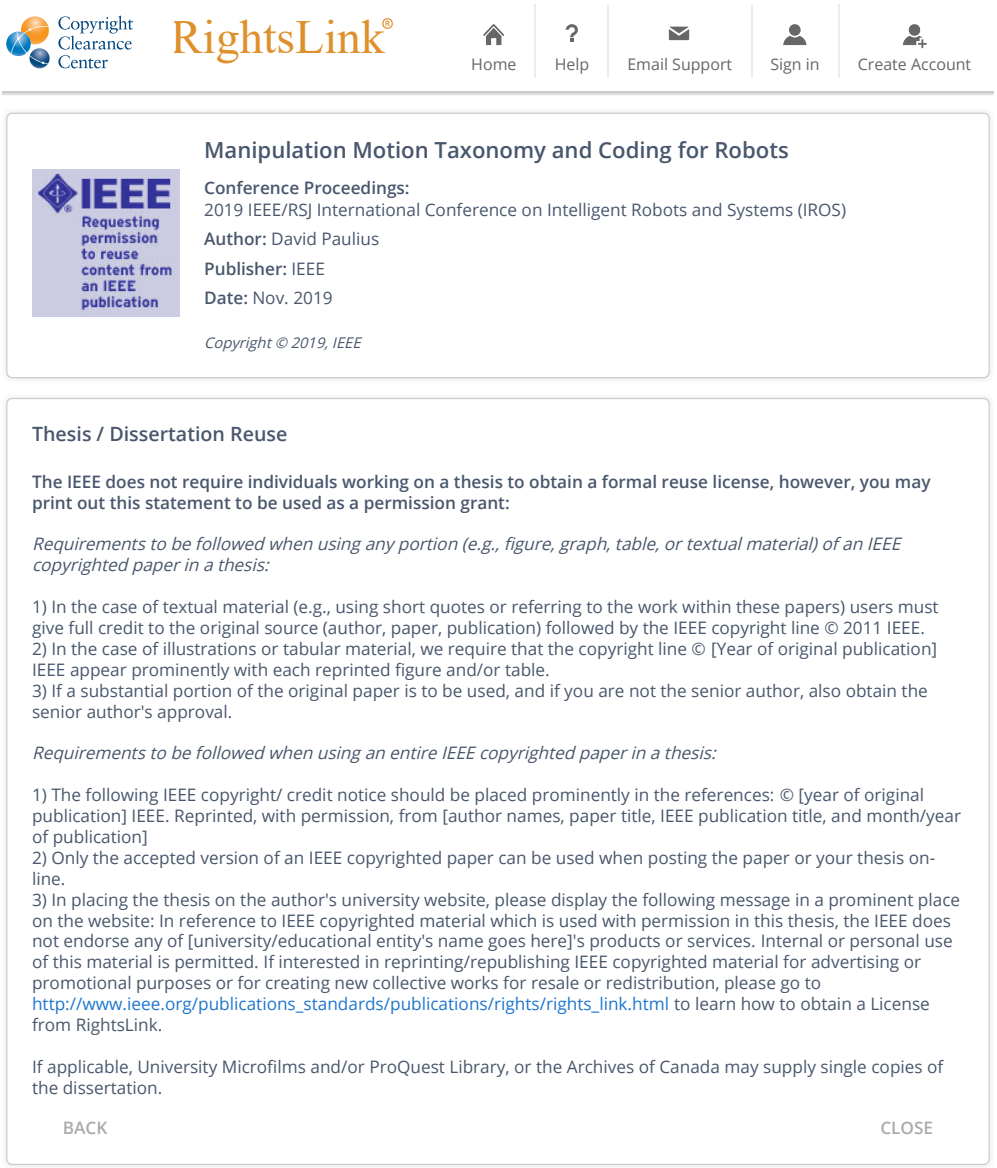
Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK | CLOSE WINDOW

Permission from [114] for use of content in Chapter 4 is shown below.



Copyright Clearance Center RightsLink®

Home | Help | Email Support | Sign in | Create Account

IEEE Requesting permission to reuse content from an IEEE publication

Manipulation Motion Taxonomy and Coding for Robots

Conference Proceedings:
2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Author: David Paulius

Publisher: IEEE

Date: Nov. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE

About the Author

Born in Caracas, Venezuela but raised in the small twin island federation of St. Kitts and Nevis, David is an aspiring computer scientist who received his Bachelor's of Science Degree in Computer Science from the University of the Virgin Islands (UVI), located in St. Thomas, VI. During his undergraduate studies, he was introduced and exposed to research with computing solutions. His two advisors, Dr. Marc Boumedine (his major advisor) and Dr. Wayne E. Archibald (his mentor and research supervisor), were pivotal in his development at UVI and motivated him to pursue further studies and to acquire his doctorate. Although his research experience at UVI pertained to computational chemistry problems (from which he obtained his first journal publication), he was generally interested in tackling practical, every day problems using computers, programming and algorithms. He visited Carnegie Mellon University as part of their Fusion Forum program and learned more about the fields of AI and robotics, and he became fascinated by the potential impacts that could be developed through these research areas. Under the supervision and mentorship of Dr. Yu Sun, David dove into robotics and enjoyed every step of the way in learning about robotics. Through hard work and dedication, he was fortunate to travel to several conferences to present their research work. He has now completed the requirements for the Doctor of Philosophy (Ph.D.) at the University of South Florida after years of hard work. Ultimately, David wishes to devote himself to researching interesting problems that would improve the quality of life for us humans and to carry the torch of mentorship and guidance by nurturing brilliant young minds through education for the development of his homelands and the Caribbean on a whole.