Functional Object-Oriented Network for Manipulation Learning

David Paulius, Roger Milton, Yongqiang Huang, William D. Buchanan, Jeanine Sam, and Yu Sun

***Abstract*— This paper presents a novel structured knowledge representation called functional object-oriented network (FOON) to model the connectivity of the functional-related objects and their motions in manipulation tasks. The graphical model FOON is learned by observing object state change and human manipulations with the objects. Using a well-trained FOON, robots can decipher a task goal, seek the correct objects at the desired states on which to operate, and generate a sequence of proper manipulation motions. The paper describes the FOON's structure and an approach to form a universal FOON with extracted knowledge from online instructional videos. A graph retrieval approach is presented to generate manipulation motion sequences from the FOON to achieve a desired goal. The results are demonstrated in a simulated environment.**

I. Introduction

Studies in neuroscience and cognitive science on object affordance [1] indicate that the mirror neurons in human brains congregate visual and motor responses [2], [3], [4]. 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 do not discharge in response to simply observing an object [5], [6]. Recently, Yoon et al. [7] studied the affordances associated to pairs of objects positioned for action and found an interesting so-called "paired object affordance effect." The effect was that the response time by right-handed participants was faster if the two objects were used together when the active object (supposed to be manipulated) was to the right of the other object.

Borghi et al. [8] further studied 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 the spatial context. The study results in neuroscience and cognitive science 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. A comprehensive review of models of affordances and canonical mirror neuron system can be found in [9].

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.

David Paulius, Roger Milton, Yongqiang Huang, William D. Buchanan, Jeanine Sam, and Yu Sun are with the Department of Computer Science and Engineering, University of South Florida. Roger Milton, William D. Buchanan, and Jeanine Sam are undergraduate students. (Contact email: yusun@mail.usf.edu)

Seeking an approach that can connect and model the motion and features of an object in the same framework is considered a new frontier in robotics. With the boom in learning from demonstration techniques in robotics [10], [11], [12], more and more researchers are trying to model object features, object affordance, and human action at the same time. Most of the research builds the relationship between single object features and human action or object affordance [13], [14], [15]. Several studies obtained and used object-action relation without considering many low-level object features. In [16], concrete object recognition was not considered, and objects were categorized solely according to object interaction sequences. Objects were segmented out from a number of video sequences, and an undirected semantic graph was used to represent the space interaction relationship between objects. With a sequence of graphs, their work was able to represent temporal and spatial interactions of objects in an event. With the semantic graphs, they constructed an event table and a similarity matrix, and the similarity between two sequences of object interaction events could be obtained according to the matrix. The objects could further be categorized according to their roles in the interactions, and the obtained semantic graphs might be used to represent robotic tasks. Jain et al. [17] developed symbolic planning that coupled robot control programs with statistical relational reasoners to arrange objects such as setting a dinner table by statistical relational learning. Yang et al. [18] proposed a manipulation action tree bank to represent actions of manipulations at multiple levels of abstraction.

Our recent work [19], [20] investigated object categorization and action recognition using an object-object-interaction affordance framework. We have developed an approach to capitalize on the strong relationship between paired objects and interactive motion by building an object relation model and associating it with a human action model in the human-object-object way to characterize inter-object affordance, and thus use the inter-object affordance relationship to improve object and action recognition.

Similar to the mirror neurons in human brains that congregate the visual and motor responses, a novel FOON is presented in this paper connects interactive objects with their functional motions to represent manipulation tasks. The proposed novel FOON focuses on the core of a manipulation task that is determined by both the objects' states and the objects' functional motions, which are represented in the FOON as connected nodes. The connections between them represent two-way dependencies that are the functional motions depending on the objects' states and the resulting state depending on the functional motion. The FOON provides structured knowledge about not only the objects and their states, but also about the relationship between the functional motions and the states. From a manipulation goal, the FOON can be searched to find the objects involved, their desired states, and the functional motions to achieve those states.

II. Functional Object-Oriented Network

A manipulation motion naturally involves one or several objects, and their states can be changed by the manipulating motion. For cases in which we deal with only one object and change its state, a simple hidden Markov model (HMM) was traditionally used to model the state change of the object. For example, a box may be "open" or "closed" as its state, and our close or open manipulation motion can change the box's state. To include the functional motion of the object, the simple HMM is modified to a more general graphical model illustrated in Figure 1(a), in which the nodes of the object states and the functional motions can be connected. In general, a single object i at its state O_i^p changes to O_i^q by a functional object motion M_k. The object states can be observed through E_i^p and E_i^q, and the functional motion can be observed through E_k^M.

In other tasks, we will manipulate one object and interact with a second object. For example, a teapot may have three different states for holding the teapot, as illustrated in Figure 2(a) to pour water into a cup, which also has three different states (Figure 2(b)). After the pouring manipulation motion, the states of both objects–the teapot and the cup–changed. Two HMMs can be structured to model the state change of the two objects separately. However, if we only use the two HMMs, the cause of the state change and the connection of the two objects are neglected.

Instead, we propose to connect the two models with a functional object motion. As illustrated in Figure 1(b), two objects i and j at their states O_i^p and O_j^u can change to their other states O_i^q and O_j^v through a manipulation reflected by a functional object motion M_k. All variables, states of the objects, and functional object motion could be hidden states with observations E_i^p, E_j^u, and E_k^M, as indicated in the figure. Pouring water from a teapot to a cup can be modeled with this model.

Progressing from the two-object cases, many objects can be connected through functional motions to form a large functional object-oriented network (FOON). Figure 3 illustrates an example FOON for a simple kitchen application. For clarity, we leave out the observation nodes. Assume we have observed a process of making a cup of sugar water for someone. With full labeling, the process can be expressed with the graphical model in Figure 3. Sugar is put in a cup with a placing motion, resulting in a cup with sugar in it. Then, a teapot with water is held to pour water into the cup with sugar with a pouring motion, resulting in a cup with water and sugar in it, and an empty teapot. At the end, a spoon is put into the cup to stir the water to help the sugar dissolve with a stirring motion, resulting in a cup of sugar water and a spoon that needs to be washed.

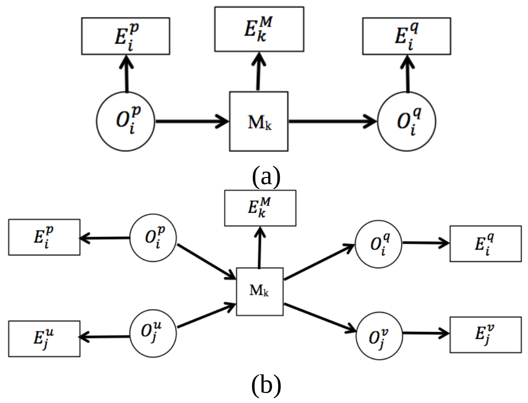

Fig. 1: (a) Graphical model for single object manipulation; (b) Graphical model for paired-objects manipulation.

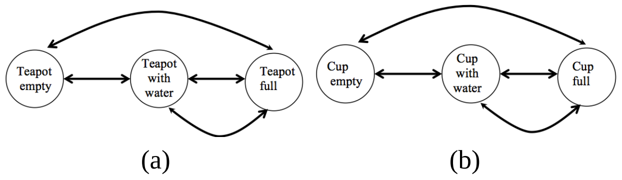

Fig. 2: (a) The states of a teapot is modeled with a state machine; (b) The states of a cup is modeled with a state machine.

A. Basic structure of FOON

Similar to a regular directed graph, a FOON contains nodes and edges. Therefore, a FOON F=(N, E) comprises a set N of nodes together with a set E of edges, where the edges have a direction associated with them.

1) Nodes: Different from a regular graph, the nodes in a FOON may have two types: object *O* or motion *M*. In a manipulation task, an object node N_O represents an object that a manipulator is holding and manipulating or an object with which the manipulator interacts. For example, in a cooking task, a person cuts bread with a knife. Here both the bread and the knife are objects and they are represented as object nodes. More broadly, any ingredient, utensil or any item in general needed to accomplish a certain task could be an object node.

In addition to the object's name, each object has a set of attributes: object state denoting its state or condition the object is observed in, and object descriptor describing whether the object is *in motion* or *stationary* in a given action. A motion node N_M describes the action that is performed on an object. A motion has only one attribute that is a motion type descriptor.

In a FOON, no two object nodes are the same and each object node in the graph is unique in terms of its name and attributes. However, a motion node of the exact same type could appear at multiple locations in the graph.

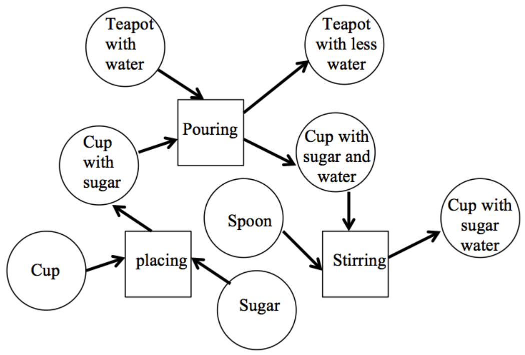

Fig. 3: An example FOON built from observing a task of making a cup of sugar water. It is composed of eleven nodes: eight object state nodes and three functional object motion nodes. For clarity, observation nodes are not shown.

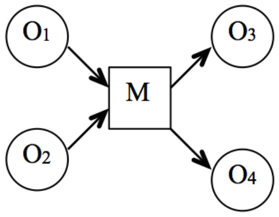

Fig. 4: Functional unit.

2) Edges: A FOON is a directed graph, since some nodes are the outcomes of certain other nodes. An edge is denoted by E that connects two nodes. Edges are drawn from either an object node to a motion node, or vice-versa, but *no two objects or two motions are directly connected to each other*. In addition, if several object nodes are connected with a motion node with edges directed to the motion node, it indicates that the objects are interacting with the motion. If a motion node has edges directed to object nodes, it indicates that the objects are the outcomes of the motion.

A FOON may be a directed acyclic graph, as there may be some instances of loops when an object does not change states from taking part in some action. A motion does not necessarily cause a change in an object, as certain objects will remain in the same state.

B. Functional unit

A functional unit is considered as the minimum learning unit in a FOON. It represents the relationship between one or several objects and *one* functional motion associated to the objects, which will be learned to represent one activity. In the functional unit shown in Figure 4, the object nodes connected with the edges pointing to the functional motion node are called incoming object nodes, while the object nodes connected with the edges pointing from the functional motion node are called output object nodes. For example, in Figure 3, the pouring motion connecting a teapot with water and a cup with sugar and resulting in a cup with water and sugar in it, and an empty teapot is one functional unit in the FOON.

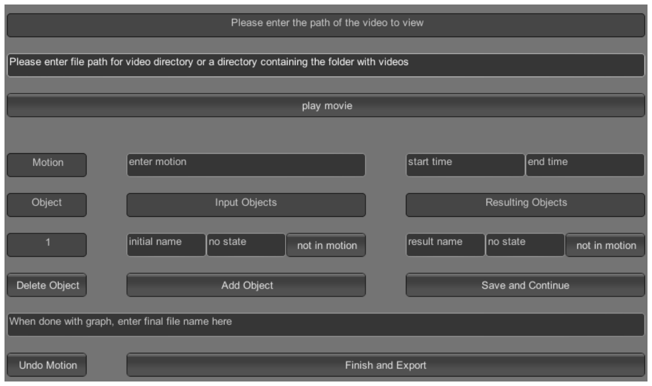

Fig. 5: The annotating interface developed tor participants to manually generate functional units of FOON.

C. Network data structure

A FOON can be represented by conventional graph representations, namely *adjacency matrices* or *adjacency lists*. We use an adjacency matrix to represent the network for its simplicity in representing a digraph. Each node is represented by a row, and its relation to other nodes is given by the columns of the matrix. An edge is denoted by a value of 1 for a given index; 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 objects and/or motion nodes found in the graph. This list is needed to map each node to its row/column representation.

In addition to the network structure, the object name and their attributes and motion types are saved and indexed with the node numbers in another file.

III. Forming FOON

Ideally, a FOON can be automatically trained from observing human activities. However, due to the complexity of the object recognition, state recognition, and motion recognition, we currently utilize human inputs to construct many small FOON subgraphs by watching instructional videos and then merge them together automatically into a universal FOON.

A. Constructing subgraphs from videos

We have developed a simple annotating interface (shown in Figure 5) for participants to watch instructional videos and input their observations one functional unit at a time. When a participant sees a motion being carried out, the participant pauses the video and inputs the motion type, incoming and outcome object names, and their states in the interface. After a motion is annotated, its functional unit will be recorded and connected to the previous functional units and form a subgraph. The subgraph will be saved into a file upon completion of the video. Each subgraph has two files: one contains labels for objects and motions and the other contains the graph representation.

The generated subgraph is then visualized and verified in a simulation environment made with Unity. Each edge is color-coded to reflect the object-motion description assigned to an object node. In the interface, if the motion description is in-motion, the color would be red; if description is not in-motion, the edge would be green.

Using the interface, we have annotated over 60 instructional videos. The full list of the videos and their FOON subgraphs can be found at [21]. Here we present one FOON subgraph of preparing garlic bread as an example. The source video is from Youtube [22]. The subgraph provides the essential structured knowledge needed to prepare the dish including objects (ingredients and utensils), their states, and their interactive motions.

B. Merging subgraphs

Since many objects appear in different subgraphs, they can be combined to form a universal FOON. However, merging of subgraphs is not as straight forward as performing a union operation since the subgraphs are made separately and with different labels. Therefore, before performing the union operation, a parser was developed to preprocess all subgraph files.

The parser has three main functions: create a main index with a list of all the objects, recreate the input file by relabeling all of the objects so they are consistent throughout, and creating a records file that records all changes in any modified files. To keep track of all data elements, there is a record class that contains object name, its old identifier, its new identifier, initial state, final state, file name, and motion. When the program is first started, it populates new lists with stored information. The program then runs the input file through a while-loop. This loop recreates the input file with new object identifiers and it extracts all of the objects, object identifiers, object states, and object motions. The extracted information is then placed into a record instance. The loop will terminate once the entire file is parsed. The new information will go through a series of for-loops and if-statements to eliminate any duplicates. When this is complete, the program rewrites the main index, state index, motion index, and record list with the new information.

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. The universal FOON graph starts from empty. The union operator first checks if objects in the subgraph already exist in the universal FOON graph and then checks if the edges are connected to the same motion nodes. If a node is completely new to the universal FOON, the node and its edges are added. The following algorithm describes the merging procedure.

Algorithm 1 Merging graph G_{new} with FOON

```
1: for all edges $E_i$ in $G_{new}$ do
2:     let $E_i$ = { $N_1$, $N_2$ }
3:     if (!$N_1$ OR $N_2$ exists in FOON) then
4:         Add node(s) to list $G$
5:     end if
6:     Create edge from node $N_1$ to $N_2$ in FOON
7: end for
```

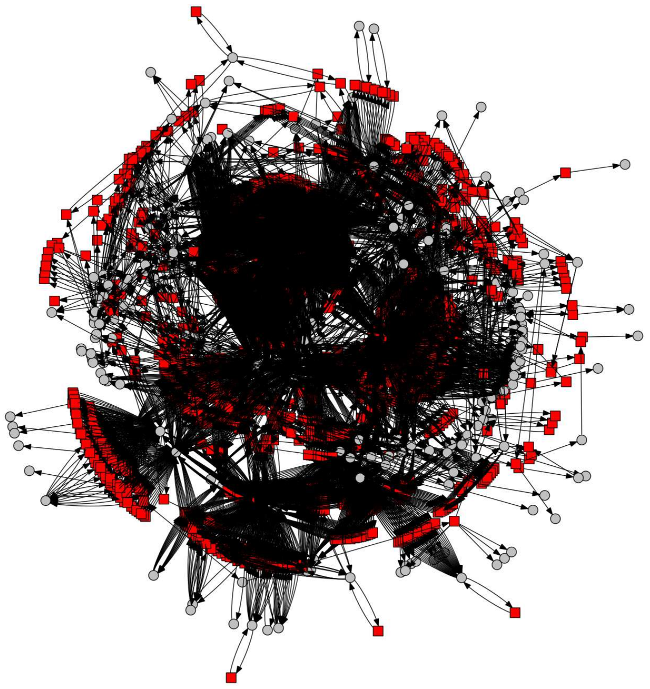

Fig. 7: Our current universal FOON that is constructed from 60 videos.

The universal FOON can be expanded by merging new subgraphs when more annotations or learning are performed. Presently, our universal FOON has ??? nodes and ??? edges after merging 60 subgraphs. However, the numbers gradually increase as more subgraphs are continuously generated.

IV. FOON Applications

The universal FOON can be considered as a structured knowledge representation that can be used for various purposes. Here we provide one example in robotics.

Considering the simple FOON example in Figure 3, a robot is assigned a task to make a cup of sugar water. With the example FOON, the robot will search it and backtrack all the required objects. First, the robot will search and find the goal state “cup of sugar water” (marked in the red circle) in the FOON. Then, using the FOON, the robot will know to look for a spoon (blue circle) and a cup with sugar and water (blue circle). If a cup with sugar and water is not available, the robot backtracks the FOON to determine what is needed to make one. As illustrated in Figure **??**, the robot knows to look for a teapot with water (green circle) and a cup with sugar (green circle). Again, if a cup with sugar is not available, from the FOON, the robot knows to look for a cup (orange circle) and sugar (orange circle). After allocating all the required objects and the connoted manipulations, the robot can perform the task with the series of manipulations learned in FOON to make a cup of sugar water.

Given a desired goal and a set of available objects (such as ingredients and tools), formally, there are two steps in generating manipulations from the FOON: retrieving a task tree and generating a task sequence. The approach is a

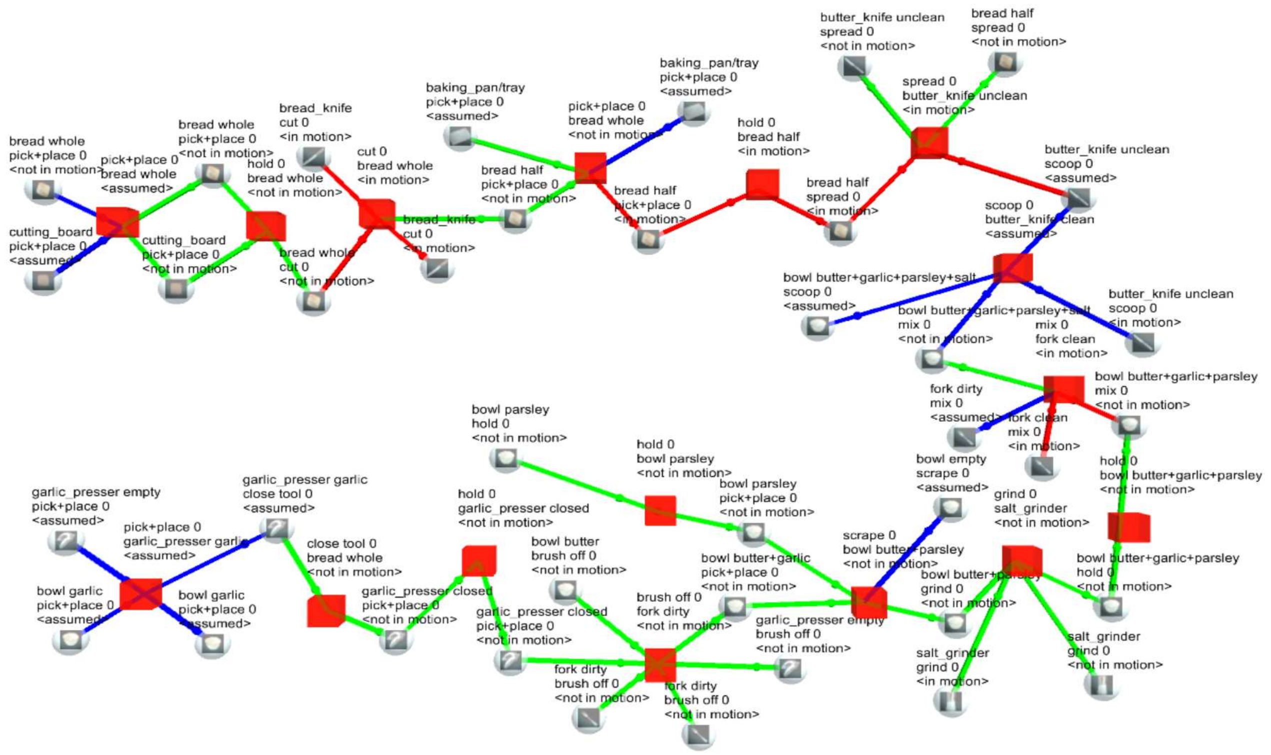

Fig. 6: A subgraph FOON generated from a making garlic bread video. (what is the color for)

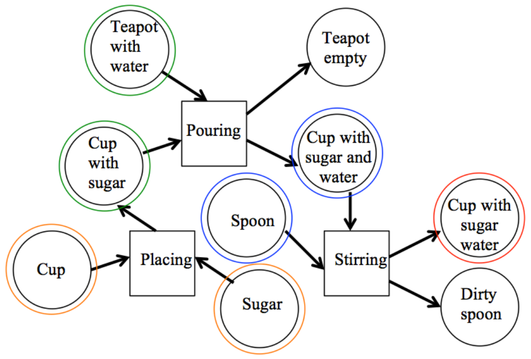

Fig. 8: Using the example FOON, a robot can look up the right objects at their desired states and a series of proper functional motions.

combination of the breadth-first search and the depth-first search.

A. Retrieving task tree

First, a goal node N_G in the FOON is identified based on the desired goal by searching the node list. The initial task tree only contains the goal node, $T = (N_G,)$. Then for each iteration, our algorithm will check if the current node is an object node or a motion node. If it is an object node, the object is checked to see if it exists in the environment and available for manipulation use (by a vision system for example, and we call the node available). If it exists, its child nodes will not be visited. If not, the algorithm will continue

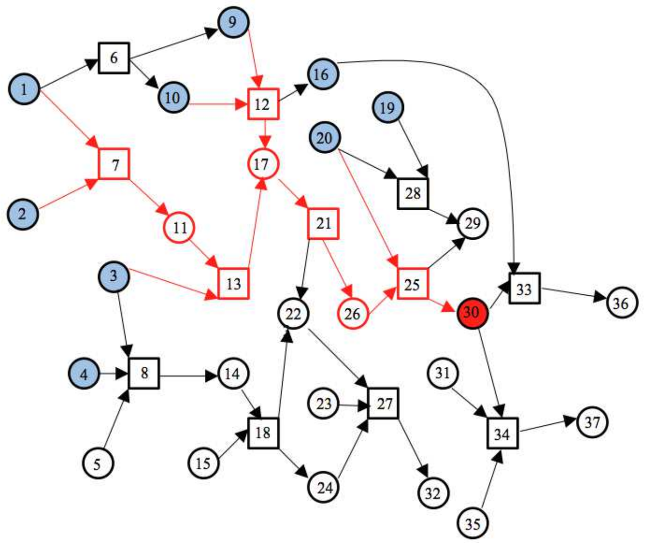

Fig. 9: An example FOON to illustrate our retrieving approach.

to visit its child nodes with depth-first search algorithm. The depth-first search algorithm is performed since there could be multiple paths to produce that object and state.

If the current node is a motion mode, the breadth-first search algorithm is performed and all the child nodes will be visited. The breadth-first search algorithm will be used since all incoming object nodes need to be present to carry out the interactive motion. The algorithm stops when all current nodes are found in the environment. The visited nodes are stored as the task tree for the desired goal.

The algorithm is illustrated in Figure 9. The available nodes are solid blue and the goal node is the solid red node with label 30. With the breadth-first search algorithm, the node 30's immediate child node 25 is added into the task tree (30, 25), and then both 25's child nodes are visited and added. The tree becomes (30, 25, 20, 26). Since node 20 is an available node, there is no need to further explore 20's child nodes. We add 26's child node 21, and then 21's child node 17. 17 is an object node and has two child nodes 12 and 13. Switching to depth first search, 12 is added to the tree and it become (30, 25, 20, 26, 21, 17, 12). From 12, we switch back to breadth-first search and add its two child nodes and the tree now is (30, 25, 20, 26, 21, 17, 12, 9, 10). Since all the current nodes are available, the algorithm stops. The task tree 1 is (30, 25, 20, 26, 21, 17, 12, 9, 10).

Alternatively, if we decide to modify the algorithm and allow exploring the 17's second child node 13, we will generate a slightly different tree. From 13, we switch back to breadth first search and add its two child nodes and the tree is (30, 25, 20, 26, 21, 17, 13, 3, 11). Since node 3 is available, we only need to go to 11's child node 7 and then reach available nodes 1 and 2. So for this option, the task tree 2 is (30, 25, 20, 26, 21, 17, 13, 3, 11, 7, 1, 2).

Since there are 3 motion nodes in the task tree 1, it would take 3 functional units to achieve the desired goal. While, there are 4 motion nodes in the second task tree and it would take 4 functional units to achieve the same goal. If all action units have the same cost, task tree 1 is more efficient than the task tree 2.

For object nodes with more than two child nodes, we perform the Branch-and-Bound algorithm that keeps track of the depth of the tree as its *cost*. Once a successful task tree is achieved, the *cost* is as the initial *best cost*. At each step of the depth-first search, the algorithm compares the current *cost* to the *best cost*, if it exceeds the *best cost*, the algorithm stops the search of that branch. If a less *costly* task tree is found, the algorithm updates the *best cost*.

B. Generating task sequence

The task tree is then used to generate a task sequence that contains a series of motions, objects, and their states, which provides step-by-step commands to a robot. For example, from the task tree 1, if it is stored in a stack, nodes 10, 9, 12, and 17 are popped out as one functional unit. The robot is instructed to operate on objects 9 and 10 with motion 12 to produce object 17. Once object 17 is produced, motion 21 is popped out with the next goal node 26 to instruct the robot to operate on object 17 to produce object 26. When object 26 is materialized, object node 20, motion node 25, and goal node 30 are popped out and the robot is instructed to operate on object 20 and 26 with motion 25 to produce object 30 that is the final desired object.

One example task tree we obtained by searching the universal FOON is shown in Figure 10. The goal is to make a cup of tea that is the red node with a set of available objects: empty kettle, water faucet, oven, empty tea cup, and tea-bag, marked as blue nodes.

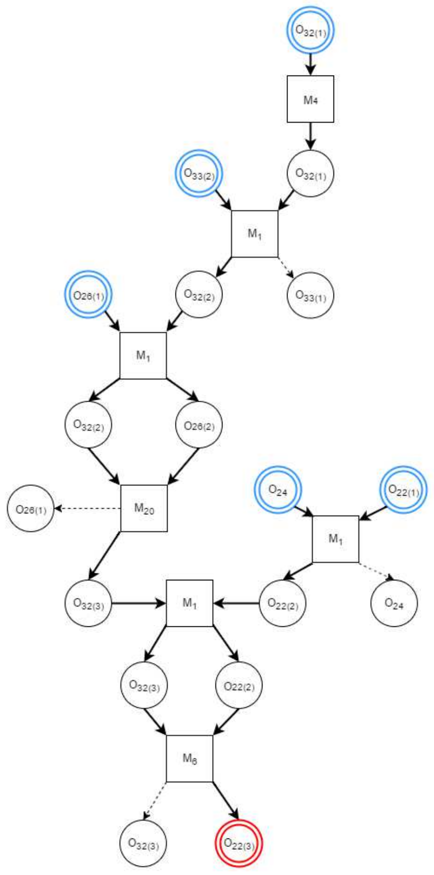

Fig. 10: One example task tree obtained by searching the universal FOON.

The following two tables provide the meanings of the labels in the task tree.

List of Objects		
Object Name	Object Type	Object State
tea cup	$O_{22(1)}$	tea cup empty
	$O_{22(2)}$	cup with tea-bag
	$O_{22(3)}$	tea
tea-bag	O_{24}	-
oven	$O_{26(1)}$	off
	$O_{26(2)}$	on
kettle	$O_{32(1)}$	empty
	$O_{32(2)}$	with water
	$O_{32(3)}$	boiling water
water faucet	$O_{33(1)}$	off
	$O_{33(2)}$	on

List of Motions	
Motion Name	Motion Type
pick-and-place	M_1
hold	M_4
pour	M_6
boil	M_{20}

In reality, more information such as object model, state

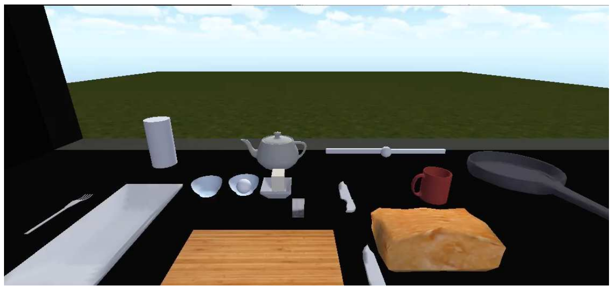
Fig. 11: A simulated kitchen environment that is used to demonstrated the generated task sequences.

model, and motion models are associated to the nodes, which allow the robot to carry out the tasks in a physical world.

C. Simulation

We have developed a simulated kitchen environment using Unity to demonstrate the usage of the task sequences. The simulation starts out with many common kitchen objects within a 3D space that are able to be moved based on input. The objects can be manipulated in several ways for the motions that the simulation is capable of and therefore creating almost endless possibilities and allowing many recipes to be done. For instance, a pick-and-place motion can be used to move bread around in space until it reaches the object that the input calls the bread to be placed on. Afterwards, a cut motion can be used with the bread with a motion object, cutting it in half and causing the state of the bread to change to halved and the knife to retain its state because the knife is not significantly affected by the bread.

Roger: How the sequence is associated to the objects in the simulation?

Simulation videos are attached.

V. Conclusion and Future Work

In this paper, we present a FOON representation of manipulation tasks, which connects interactive objects with their functional motions. The FOON provides structured knowledge about the relationship between the object states and functional object motions, which is valuable for not only learning manipulation tasks, but also understanding human activities.

We have developed an approach to construct functional units using abstracted knowledge of online instructional videos, mainly cooking videos. The functional units are then connected into subgraphs and then merged into a universal FOON. Manipulation knowledge can be retrieved from the FOON based on a manipulation goal using our algorithms. The manipulation knowledge is stored in a task tree and then converted into a task sequence with a series of involved objects, manipulation motions, and immediate goals.

A simulated kitchen environment has been developed to demonstrate the usage of the task sequences generated from the universal FOON. Two demo videos are attached with this paper and additional demo videos are available at [21].

In the future, we plan to develop better representations of manipulation motions based on our motion harmonics representation [23] than the currently-used manipulation motion names. In addition to motion, other aspects of the task such as object recognition and task-based grasping [24], [25] will be considered in more details.

The current data structure of the network is not ideal for sparse graphs, which should be improved as the FOON grows. More network analyses, such as computing connection strength and efficiency, will be performed on the universal FOON to better use and understand the dynamics of the FOON.

VI. Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. 1421418.

References

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

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

[3] 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.

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

[5] 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.

[6] 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.

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

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

[9] 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.

[10] 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.

[11] B. D. Argall, S. Chernova, and et al. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[12] S. Schaal, S. Ijspeert, and et al. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London Series B- Biological Sciences*, 358(1431):537–547, 2003.

[13] A. Gupta and L. Davis. Objects in action: An approach for combining action understanding and object perception. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

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

[15] J. Gall, A. Fossati, and L Gool. Functional categorization of objects using real-time markerless motion capture. In *Conference on Computer Vision and Pattern Recognition*, pages 1969–1976, 2011.

[16] E. Aksoy, A. Abramov, F. Worgotter, and B. Dellen. Categorizing object-action relations from semantic scene graphs. In *IEEE Intl. Conference on Robotics and Automation*, pages 398–405, 2010.

[17] 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.

[18] 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.

[19] Y. Sun, S. Ren, and Y. Lin. Object-object interaction affordance learning. *Robotics and Autonomous Systems*, 2013.

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

[21] Online foon graphs and videos, http://www.foonet.com.

[22] A garlic bread video on youtube, https://www.youtube.com/watch?v=hzfoqed9pwy.

[23] Y Huang and Y. Sun. Generating manipulation trajectories using motion harmonics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, page in press, 2015.

[24] Y. Lin and Y. Sun. Robot grasp planning based on demonstrated grasp strategies (in press),. *Intl. Journal of Robotics Research*, 34(1):26–42, 2015.

[25] Y. Lin and Y. Sun. Grasp planning to maximize task coverage. *Intl. Journal of Robotics Research*, 34(9):1195–1210, 2015.